

Games for Active XML Revisited

Martin Schuster Thomas Schwentick
TU Dortmund University

Friday 19th December, 2014

Abstract

The paper studies the rewriting mechanisms for intensional documents in the Active XML framework, abstracted in the form of *active context-free games*. The *safe rewriting* problem studied in this paper is to decide whether the first player, JULIET, has a winning strategy for a given game and (nested) word; this corresponds to a successful rewriting strategy for a given intensional document. The paper examines several extensions to active context-free games.

The primary extension allows more expressive schemas (namely XML schemas and regular nested word languages) for both target and replacement languages and has the effect that games are played on nested words instead of (flat) words as in previous studies. Other extensions consider validation of input parameters of web services, and an alternative semantics based on insertion of service call results.

In general, the complexity of the safe rewriting problem is highly intractable (doubly exponential time), but the paper identifies interesting tractable cases.

1 Introduction

Scientific context This paper contributes to the theoretical foundations of intensional documents, in the framework of *Active XML* [1]. It studies game-based abstractions of the mechanism transforming intensional documents into documents of a desired form by calling web services. One form of such games has been introduced under the name *active context-free games* in [14] as an abstraction of a problem studied in [12].¹ The setting in [12] is as follows: an *Active XML* document is given, where some elements consist of functions representing web services that can be called. The goal is to rewrite the document by a series of web service calls into a document matching a given *target schema*.

Towards an intuition of Active XML document rewriting, consider the example in Figure 1 of an online local news site dynamically loading information about weather and local events (adapted from [12] and [14]). Figure 1a shows the initial Active XML document for such a site, containing *function nodes* which refer to a weather and an event service, respectively, instead of concrete weather and event data. After a single function call to each of these services has been materialised, the resulting document may look like the one depicted

¹Actually, the two notions were introduced in the respective conference papers.

in Figure 1b. Note that the rewritten document now contains new function nodes; further rewriting might be necessary to reach a document in a given target schema (which could, for instance, require that the document contains at least one indoor event if the weather is rainy).

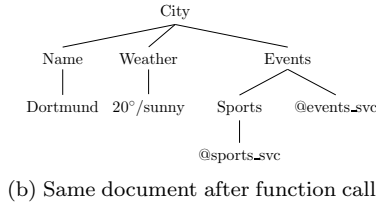
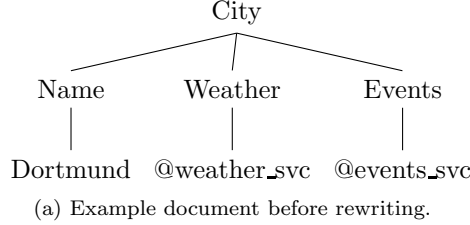


Figure 1: Example of Active XML rewriting.

Modelling this rewriting problem as a game follows the approach of dealing with uncertainty by playing a “game against nature”: We model the process intended to rewrite a given document into a target schema by performing function calls as a player (JULIET). As her moves, she chooses which function nodes to call, and her goal is to reach a document in the target schema. Returns of function calls, on the other hand, are chosen (in accordance with some schema for each called service) by an antagonistic second player (ROMEO), whose goal is to foil JULIET. The question whether a given document can always be rewritten into the target schema may then be solved by deciding whether JULIET has a winning strategy. More specifically, given an input document, target schema and return schemas for function calls, there should exist a *safe rewriting* algorithm that always rewrites the input document into the target schema, no matter the concrete returns of function calls, if and only if JULIET has a winning strategy in the corresponding game.²

In [12], the target schema is represented by an XML document type definition (DTD). It was argued that, due to the restricted nature of DTDs, the problem can be reduced to a rewriting game on strings where, in each move a single symbol is replaced by a string, the set of allowed replacement strings for each symbol is a regular language and the target language is regular³, as well.

In [14] the complexity of the problem to determine the winner in such games (mainly with finite replacement languages) was studied. Whereas this problem is undecidable in general, there are important cases in which it can be solved, particularly if JULIET chooses the symbols to be replaced in a left-to-right fashion. In and after [14, 12], research very much concentrated on games on strings (and thus on the setting with DTDs). Furthermore, to achieve tractability, a

²It is hard to give a precise statement of *safe rewriting* that does not already involve games, but we hope that the general idea of this statement becomes sufficiently clear.

³More precisely, it should be given by a *deterministic* regular expression.

	No replay	Bounded	Unbounded
Regular target language			
Regular replacement	PSPACE	2-EXPTIME	2-EXPTIME
Finite replacement	PSPACE	PSPACE	EXPTIME
DTD or XML Schema target language			
Regular replacement	PTIME	PSPACE	EXPTIME
Finite replacement	PTIME	PTIME	EXPTIME

Table 1: Summary of complexity results. All results are completeness results.

special emphasis was given to the restriction to *bounded* strategies, in which the recursion depth with respect to web service calls is bounded by some constant.

Our approach The aim of this paper is to broaden the scope and extend the investigation of games for Active XML in several aspects. First of all, we consider stronger schema languages (compared to DTDs) such as XML Schema and Relax NG, due to their practical importance. To allow for this extension, our games are played on nested words [3].⁴

Furthermore, we study the impact of the validation of input parameters for web service calls (partly considered already in [12]), and investigate an alternative semantics, where results of web service calls are inserted next to the node representing the web service, as opposed to replacing that node.

As we are particularly interested in the identification of tractable cases, we follow the previous line of research by concentrating on strategies in document order (left-to-right strategies) and by considering bounded strategies (*bounded replay*) and strategies in which no calls in results from previous web service calls are allowed (*no replay*). However, we also pinpoint the complexity of the general setting.

As a basic intuition for the concept of replay, consider again the online news site example from Figure 1, and assume that the schema for the event service’s returns is (partially) given by $\text{@event_svc} \rightarrow (\text{Sports|Movie})\text{@event_svc}$, i.e. the event service allows for dynamic loading of additional results. A strategy with no replay would not be allowed to fetch any additional results in the situation of Figure 1b, while a strategy with bounded replay k (for some constant k) could load up to k more events after the first. A strategy with unbounded replay would be able to fetch an arbitrary number of results, but might lead to a rewriting process that does not terminate if unsuccessful.

Our contributions Our complexity results with respect to stronger schema languages are summarised in Table 1. In the general setting, the complexity is very bad: doubly exponential time. However, there are tractable cases for XML Schema: replay-free strategies in general and strategies with bounded replay in the case of finite replacement languages (that is, when there are only finitely many possible answers, for each web service). It should be noted that the PSPACE-hardness result for the case with DTDs, bounded replay and infinite replacement languages indicates that the respective PTIME claim in [12] is wrong.

In the setting where web services come with an input schema that restricts the parameters of web service calls, we only study replay-free strategies. It

⁴More precisely: word encodings of nested words in the sense of [3].

turns out that this case is tractable if all schemas are specified by DTDs and the number of web services is bounded. On the other hand, if the desired document structure is specified by an XML Schema or the number of function symbols is unbounded, the task becomes PSPACE-hard.

For insertion-based semantics, we identify an undecidable setting and establish a correspondence with the standard “replacement” semantics, otherwise.

As a side result of independent interest, we show that the word problem for alternating nested word automata is PSPACE-complete.

Related Work We note that the results on flat strings in this paper do not directly follow from the results in [14], as [14] assumed target languages given by DFAs as opposed to *deterministic* regular expressions, which are integral to both DTDs and more expressive XML schema languages. However, the techniques from [14] can be adapted.

More related work for active context-free games than the papers mentioned so far is discussed in [14]. Further results on active context-free games in the “flat strings” setting can be found in [2, 4]. A different form of 2-player rewrite games are studied in [18]. More general *structure rewriting games* are defined in [9].

Organisation We give basic definitions in Section 2. Games with regular schema languages (given by nested word automata) are studied in Section 3, games in which the schemas are given as DTDs or XML Schemas are investigated in Section 4. Validation of parameters and insertion of web service results are considered in Section 5. Most proofs are delegated to the appendix for brevity.

Acknowledgements We would like to thank the anonymous reviewers for their insightful and constructive comments. We are grateful to Nils Vortmeier and Thomas Zeume for careful proof reading, and to Krystian Kensy for checking our proof of Proposition 18 (b) and for pinpointing the problems in the algorithm of [12] as part of his Master’s thesis.

2 Preliminaries

For any natural number $n \in \mathbb{N}$, we denote by $[n]$ the set $\{1, \dots, n\}$. Where M is a (finite) set, $\mathcal{P}(M)$ denotes the powerset of M , i.e. the set of all subsets of M . For an alphabet Σ , we denote the set of finite strings over Σ by Σ^* and ϵ denotes the empty string.

Nested words We use nested words⁵ as an abstraction of XML documents [3]. For a finite alphabet Σ , $\langle \Sigma \rangle \stackrel{\text{def}}{=} \{\langle a \rangle \mid a \in \Sigma\}$ denotes the set of all *opening* Σ -tags and $\langle / \Sigma \rangle \stackrel{\text{def}}{=} \{\langle / a \rangle \mid a \in \Sigma\}$ the set of all *closing* Σ -tags. The set $\text{WF}(\Sigma) \subseteq (\langle \Sigma \rangle \cup \langle / \Sigma \rangle)^*$ of *(well-)nested words over* Σ is the smallest set such that $\epsilon \in \text{WF}(\Sigma)$, and if $u, v \in \text{WF}(\Sigma)$ and $a \in \Sigma$, then also $u\langle a \rangle v\langle / a \rangle \in \text{WF}(\Sigma)$. We (informally) associate with every nested word w its *canonical forest representation*, such that words $\langle a \rangle \langle / a \rangle$, $\langle a \rangle v \langle / a \rangle$ and uv correspond to an a -labelled leaf, a tree with root

⁵Our definition of nested words corresponds to word encodings of well-matched nested words in [3].

a (and subforest corresponding to v), and the forest of u followed by the forest of v , respectively. A nested string w is *rooted*, if its corresponding forest is a tree. In a nested string $w = w_1 \dots w_n \in \text{WF}(\Sigma)$, two tags $w_i \in \langle \Sigma \rangle$ and $w_j \in \langle / \Sigma \rangle$ with $i < j$ are *associated* if the substring $w_i \dots w_j$ of w is rooted. To stress the distinction from nested strings in $\text{WF}(\Sigma)$, we refer to strings in Σ^* as *flat strings* (over Σ).

What we describe as opening and closing tags is often referred to as *call symbols* and *return symbols* in the literature on nested words; we avoid these terms to avoid confusion with Read and Call moves used in context-free games (see below).

Context-free games A *context-free game on nested words* (cfG) $G = (\Sigma, \Gamma, R, T)$ consists⁶ of a finite alphabet Σ , a set $\Gamma \subseteq \Sigma$ of *function symbols*, a *rule set* $R \subseteq \Gamma \times \text{WF}(\Sigma)$ and a *target language* $T \subseteq \text{WF}(\Sigma)$. We will only consider the case where T and, for each symbol $a \in \Gamma$, the set $R_a \stackrel{\text{def}}{=} \{u \mid (a, u) \in R\}$ is a non-empty regular nested word language, to be defined in the next subsection.

A play of G is played by two players, JULIET and ROMEO, on a word $w \in \text{WF}(\Sigma)$. In a nutshell, JULIET moves the focus along w in a left-to-right manner and decides, for every closing tag⁷ $\langle /a \rangle$ whether she plays a *Read* or, in case $a \in \Gamma$, a *Call* move. In the latter case, ROMEO then replaces the rooted word ending at the position of $\langle /a \rangle$ with some word $v \in R_a$ and the focus is set on the first symbol of v . In case of a Read move (or an opening tag) the focus just moves further on. JULIET wins a play if the word obtained at its end is in T .

Towards a formal definition, a *configuration* is a tuple $\kappa = (p, u, v) \in \{J, R\} \times ((\Sigma) \cup \langle / \Sigma \rangle)^* \times ((\Sigma) \cup \langle / \Sigma \rangle)^*$ where p is the player to move, $uv \in \text{WF}(\Sigma)$ is the *current word*, and the first symbol of v is the *current position*. A *winning configuration* for JULIET is a configuration $\kappa = (J, u, \epsilon)$ with $u \in T$. The configuration $\kappa' = (p', u', v')$ is a *successor configuration* of $\kappa = (p, u, v)$ (Notation: $\kappa \rightarrow \kappa'$) if one of the following holds:

- (1) $p' = p = J$, $u' = us$, and $sv' = v$ for some $s \in \langle \Sigma \rangle \cup \langle / \Sigma \rangle$ (JULIET plays Read);
- (2) $p = J$, $p' = R$, $u = u'$, $v = v' = \langle /a \rangle z$ for $z \in ((\Sigma) \cup \langle / \Sigma \rangle)^*$, $a \in \Gamma$, (JULIET plays Call);
- (3) $p = R$, $p' = J$, $u = x \langle a \rangle y$, $v = \langle /a \rangle z$ for $x, z \in ((\Sigma) \cup \langle / \Sigma \rangle)^*$, $y \in \text{WF}(\Sigma)$, $u' = x$ and $v' = y'z$ for some $y' \in R_a$ (ROMEO plays y');⁸

The *initial configuration* of game G for string w is $\kappa_0(w) \stackrel{\text{def}}{=} (J, \epsilon, w)$. A *play* of G is either an infinite sequence $\Pi = \kappa_0, \kappa_1, \dots$ or a finite sequence $\Pi = \kappa_0, \kappa_1, \dots, \kappa_k$ of configurations, where, for each $i > 0$, $\kappa_{i-1} \rightarrow \kappa_i$ and, in

⁶Some of the following definitions are taken from [4].

⁷It is easy to see that the winning chances of the game do not change if we allow JULIET to play Call moves at opening tags: if JULIET wants to play Call at an opening tag she can simply play Read until the focus reaches the corresponding closing tag and play Call then. On the other hand, if she can win a game by calling a closing tag, she can also win it by calling the corresponding opening tag, thanks to the fact that she has full information.

⁸We note that a Call move on $\langle /a \rangle$ in a substring of the form $\langle a \rangle y \langle /a \rangle$ actually deletes the substring y along with the opening and closing a -tags. This is consistent with the AXML intuition of the subtree rooted at a function node getting replaced when the function node is called.

the finite case, κ_k has no successor configuration. In the latter case, JULIET *wins* the play if κ_k is of the form (J, u, ϵ) with $u \in T$, in all other cases, ROMEO wins.

Strategies A *strategy* for player $p \in \{J, R\}$ maps prefixes $\kappa_0, \kappa_1, \dots, \kappa_k$ of plays, where κ_k is a p -configuration, to allowed moves. We denote strategies for JULIET by $\sigma, \sigma', \sigma_1, \dots$ and strategies for ROMEO by $\tau, \tau', \tau_1, \dots$.

A strategy σ is *memoryless* if, for every prefix $\kappa_0, \kappa_1, \dots, \kappa_k$ of a play, the selected move $\sigma(\kappa_0, \kappa_1, \dots, \kappa_k)$ only depends on κ_k . As context-free games are reachability games we only need to consider memoryless games; see, e.g., [8].

Proposition 1. Let G be a context-free game, and w a string. Then either JULIET or ROMEO has a winning strategy on w , which is actually memoryless.

Therefore, in the following, strategies σ for JULIET map configurations κ to moves $\sigma(\kappa) \in \{\text{Call}, \text{Read}\}$ and strategies τ for ROMEO map configurations κ to moves $\tau(\kappa) \in \text{WF}(\Sigma)$.

For configurations κ, κ' and strategies σ, τ we write $\kappa \xrightarrow{\sigma, \tau} \kappa'$ if κ' is the unique successor configuration of κ determined by strategies σ and τ . Given an initial word w and strategies σ, τ the play⁹ $\Pi(\sigma, \tau, w) \stackrel{\text{def}}{=} \kappa_0(w) \xrightarrow{\sigma, \tau} \kappa_1 \xrightarrow{\sigma, \tau} \dots$ is uniquely determined. If $\Pi(\sigma, \tau, w)$ is finite, we denote the word represented by its final configuration by $\text{word}_G(w, \sigma, \tau)$.

A strategy σ for JULIET is *finite* on string w if the play $\Pi(\sigma, \tau, w)$ is finite for every strategy τ of ROMEO. It is a *winning strategy* on w if JULIET wins the play $\Pi(\sigma, \tau, w)$, for every τ of ROMEO. A strategy τ for ROMEO is a *winning strategy* for w if ROMEO wins $\Pi(\sigma, \tau, w)$, for every strategy σ of JULIET. We only consider finite strategies for JULIET, due to JULIET's winning condition. We denote the set of all finite strategies for JULIET in the game G by $\text{STRAT}_J(G)$, and the set of all strategies for ROMEO by $\text{STRAT}_R(G)$.

The *Call depth* of a play Π is the maximum nesting depth of Call moves in Π , if this maximum exists. That is, the Call depth of a play is zero, if no Call is played at all, and one, if no Call is played inside a string yielded by a replacement move. For a strategy σ of JULIET and a string $w \in \text{WF}(\Sigma)$, the *Call depth* $\text{Depth}^G(\sigma, w)$ of σ on w is the maximum Call depth in any play $\Pi(\sigma, \tau, w)$. A strategy σ has *k-bounded Call depth* if $\text{Depth}^G(\sigma, w) \leq k$ for all $w \in \text{WF}(\Sigma)$. We denote by $\text{STRAT}_J^k(G)$ the set of all strategies with k -bounded Call depth for JULIET on G . As a more intuitive formulation, we use the concept of *replay*, which is defined as Call depth (if it exists) minus one: Strategies for JULIET of Call depth one are called *replay-free*, and strategies of k -bounded Call depth, for any k , have *bounded replay*. For technical reasons, we need to use Call depth for some formal proofs and definitions, but we will stick with the more intuitive concept of replay wherever possible.

By $\text{JWin}(G)$ we denote the set of all words for which JULIET has a winning strategy in $\text{STRAT}_J(G)$ (likewise for $\text{JWin}^k(G)$ and $\text{STRAT}_J^k(G)$).

Nested word automata A *nested word automaton* (NWA) $A = (Q, \Sigma, \delta, q_0, F)$ [3] is basically a pushdown automaton which performs a push operation on every

⁹As the underlying game G will always be clear from the context, our notation does not mention G explicitly.

opening tag and a pop operation on every closing tag, and in which the push-down symbols are just states. More formally, A consists of a set Q of *states*, an alphabet Σ , a *transition function* δ , an *initial state* $q_0 \in Q$ and a set $F \subseteq Q$ of *accepting states*. The function δ is the union of a function $(Q \times \langle \Sigma \rangle) \rightarrow \mathcal{P}(Q \times Q)$ and a function $(Q \times Q \times \langle \Sigma \rangle) \rightarrow \mathcal{P}(Q)$.

A *configuration* κ of A is a tuple $(q, \alpha) \in Q \times Q^*$, with a *linear state* q and a sequence α of *hierarchical states*, reflecting the pushdown store. A *run of A on $w = w_1 \dots w_n \in WF(\Sigma)$* is a sequence $\kappa_0, \dots, \kappa_n$ of configurations $\kappa_i = (q_i, \alpha_i)$ of A such that for each $i \in [n]$ and $a \in \Sigma$ it holds that

- if $w_i = \langle a \rangle$, $(q_i, p) \in \delta(q_{i-1}, \langle a \rangle)$ (for some $p \in Q$), and $\alpha_i = p\alpha_{i-1}$, or
- if $w_i = \langle /a \rangle$, $q_i \in \delta(q_{i-1}, p, \langle /a \rangle)$ (for some $p \in Q$), and $p\alpha_i = \alpha_{i-1}$.

In this case, we also write $\kappa_0 \xrightarrow{w}_A \kappa_n$. We say that A *accepts* w if $(q_0, \epsilon) \xrightarrow{w}_A (q', \epsilon)$ for some $q' \in F$. The language $L(A) \subseteq WF(\Sigma)$ is defined as the set of all strings accepted by A and is called a *regular language* (of nested words).

An NWA is *deterministic* (or DNWA) if $|\delta(q, \langle a \rangle)| = 1 = |\delta(q, p, \langle /a \rangle)|$ for all $p, q \in Q$ and $a \in \Sigma$. In this case, we simply write $\delta(q, \langle a \rangle) = (q', p')$ instead of $\delta(q, \langle a \rangle) = \{(q', p')\}$ (and accordingly for $\delta(q, p, \langle /a \rangle)$), and $\delta^*(p, w) = q$ if q is the unique state, for which $(p, \epsilon) \xrightarrow{w}_A (q, \epsilon)$.

An NWA is in *normal form* if every transition function $\delta(p, \langle a \rangle)$ only uses pairs of the form (q, p) . Informally, when A reads an opening tag it always pushes its current state (before the opening tag) and therefore can see this state when it reads the corresponding closing tag. As in this case the hierarchical state is just the origin state p of the transition, we write $\delta(p, \langle a \rangle) = q$ as an abbreviation of $\delta(p, \langle a \rangle) = (q, p)$, for DNWAs in normal form.

Lemma 2. There is a polynomial-time algorithm that computes for every deterministic NWA an equivalent deterministic NWA in normal form.

Algorithmic Problems In this paper, we study the following algorithmic problem $\text{JWIN}(\mathcal{G})$ for various classes \mathcal{G} of context-free games.

$\text{JWIN}(\mathcal{G})$	
Given:	A context-free game $G \in \mathcal{G}$ and a string w .
Question:	Is $w \in \text{JWin}(G)$?

A class \mathcal{G} of context-free games in $\text{JWIN}(\mathcal{G})$ comes with three parameters:

- the representation of the target language T ,
- the representation of the replacement languages R_a , and
- to which extent replay is restricted.

It is a fair assumption that the representations of the target language and the replacement languages are of the same kind, but we will always discuss the impact of the replacement language representations separately. In our most general setting, investigated in Section 3, target languages are represented by deterministic nested word automata, and replacement languages by (not necessarily deterministic) nested word automata. We do not consider the representation of target

languages by non-deterministic NWAs, as (1) already for DNWAs the complexity is very high in general, and (2) we can show that even in the replay-free case the complexity would become EXPTIME-complete. We usually denote the automata representing the target and replacement languages by $A(T)$ and $A(R_a)$, respectively.

In Section 4 we study the cases where T is given as an XML Schema or a DTD. In each setting, we consider the cases of unrestricted replay, bounded replay (Call depth k , for some k), and no replay (Call depth 1). We note that replay depth is formally not an actual game parameter, but the algorithmic problem can be restricted to strategies of JULIET of the stated kind.

If the class \mathcal{G} of games is clear from the context, we often simply write JWIN instead of JWIN(\mathcal{G}).

We denote by $|R|$ the combined size of all $A(R_a)$, $a \in \Gamma$, and by $|G|$ the size of (a sensible representation of) G , i.e. $|G| = |\Sigma| + |R| + |A(T)|$.

3 Games with regular target languages

We first consider our most general case, where target languages are given by DNWAs, replacement languages by NWAs and replay is unrestricted, because the algorithm that we develop for this case can be adapted (and sped up) for many of the more restricted cases. It is important to note that our results do not *rely* on the presentation of schemas as nested word automata. In fact, in Section 4, we will assume that the target schema is given as an XML Schema or a DTD. However, for our algorithms nested word automata are handy to represent (linearisations of) regular tree languages and therefore in *this* section target languages are represented by NWAs. We emphasize that deterministic bottom-up tree automata can be translated into deterministic NWAs in polynomial time [3].

This generic algorithm works in two main stages for a given cfG G and word w . It first analyses the game G and aggregates all necessary information in a so-called *call effect* C . Then it uses C to decide whether JULIET has a winning strategy in the game G on w .

The call effect C only depends on G and contains, for every function symbol f and every state q of the $A(T)$, all possible effects of the subgame starting with a Call move of JULIET on some symbol $\langle /f \rangle$ on the target language T , under the assumption that the sub-computation of $A(T)$ on the word yielded by the game from $\langle /f \rangle$ starts in state q . More precisely, it summarises which sets S of states JULIET can enforce by some strategy σ , where each S is a set of states of $A(T)$ that ROMEO might enforce with a counter strategy against σ .

The first stage of the algorithm consists of an inductive computation in which successive approximations C^1, C^2, \dots of C are computed, where C^i is the restriction of C to strategies of JULIET of Call depth i . The size of call effects and the number of iterations are at most exponential in $|G|$. However, the first stage can not be performed in exponential time as a single iteration might take doubly exponential time in $|G|$. It turns out through our corresponding lower bound that single iterations can not be done faster.

At the end of the first stage, the algorithm computes an alternating NWA A_G (of exponential size) from C that decides the set JWIN(G). In the second stage, A_G is evaluated on w , taking at most polynomial space in $|A_G|$ and $|w|$.

A restriction of games to bounded replay does not improve the general complexity of the problem, as this is dominated by the doubly exponential effort of a single iteration. However, for replay-free games, no iterations are needed, the initial call effect C^1 is of polynomial size and can easily be computed and therefore, in this case, the overall complexity is dominated by the second stage, yielding a polynomial-space algorithm.

Altogether we prove the following theorem in this section.

Theorem 3. For the class of unrestricted games $\text{JWIN}(\mathcal{G})$ is

- (a) 2-EXPTIME-complete with unbounded replay,
- (b) 2-EXPTIME-complete with bounded replay, and
- (c) PSPACE-complete without replay.

The rest of this section gives a proof sketch for Theorem 3.

Before we describe the generic algorithm in more detail, we discuss the very natural and more direct approach by alternating algorithms, in which a strategy for JULIET is nondeterministically guessed and the possible moves of ROMEO are taken care of by universal branching. In our setting of context-free games, there are the following obstacles to this approach: (1) ROMEO can, in general, choose from an infinite number of (and thus arbitrarily long) strings in R_a , for the current a , and (2) it is not a priori clear that such algorithms terminate on all branches. Whereas the latter obstacle is not too serious (if JULIET has a winning strategy, termination on all branches is guaranteed), the former requires a more refined approach. We basically deal with it in two ways: in some cases it is possible to show that it does not help ROMEO to choose strings of length beyond some bound; in the remaining cases (in particular in those cases considered in this section), the algorithms use abstracted moves instead of the actual replacement moves of the game. The two stages that were sketched above, then come very naturally: first, the abstraction has to be computed, then it can be used for the actual alternating computation.

Our abstraction from actual cfGs is based on the simple observation that instead of knowing the final word $\text{word}_G(w, \sigma, \tau)$ that is reached in a play $\Pi(\sigma, \tau, w)$, it suffices to know whether $\delta^*(q_0, \text{word}_G(w, \sigma, \tau)) \in F$ to tell the winner. If we fix a strategy σ of JULIET in a game on w , the possible outcomes of the game (for the different strategies of ROMEO) can thus be summarised by $\text{states}_G(q_0, w, \sigma) \stackrel{\text{def}}{=} \{\delta^*(q_0, \text{word}_G(w, \sigma, \tau)) \mid \tau \in \text{STRAT}_R(G)\}$.

To this end, it will be particularly useful to study the (abstractions of) possible outcomes of *subgames* that start from a Call move on some tag $\langle a \rangle$ until the focus moves to the symbol after $\langle a \rangle$.

Definition 4. For a cfG $G = (\Sigma, \Gamma, R, T)$ with a deterministic target NWA $A(T) = (Q, \Sigma, \delta, q_0, F)$, the *call effect* $\mathcal{C}[G] : \Gamma \times Q \rightarrow \mathcal{P}(\mathcal{P}(Q))$ is defined, for every $a \in \Gamma$, $q \in Q$, by

$$\mathcal{C}[G](a, q) \stackrel{\text{def}}{=} [\{\text{states}_G(q, \langle a \rangle \langle a \rangle, \sigma) \mid \sigma \in \text{STRAT}_{\text{J,Call}}(G)\}]_{\min},$$

where $\text{STRAT}_{\text{J,Call}}(G)$ contains all strategies of JULIET that start by playing Read on $\langle a \rangle$ and Call on $\langle a \rangle$, and the operator $[\cdot]_{\min}$ removes all non-minimal sets from a set of sets.

We next describe how to compute $\mathcal{C}[G]$ from a given cfG G . As already mentioned, our algorithm follows a fixpoint-based approach. It computes inductively, for $k = 1, 2, \dots$ the call effect of the restricted game of maximum Call depth k . We show that the fixpoint reached by this process is the actual call effect $\mathcal{C}[G]$.

To this end, let, for every cfG G , $a \in \Sigma$, $q \in Q$, and $k \geq 1$,

$$\mathcal{C}^k[G](a, q) \stackrel{\text{def}}{=} \left[\{ \text{states}_G(q, \langle a \rangle \langle /a \rangle, \sigma) \mid \sigma \in \text{STRAT}_{J, \text{Call}}^k(G) \} \right]_{\min}.$$

As an important special case, the call effect of replay-free games — the basis for the inductive computation — consists of only one set.

Lemma 5. For every $q \in Q$ and $a \in \Sigma$, it holds that

$$\mathcal{C}^1[G](a, q) = \{ \{ \delta^*(q, v) \mid v \in R_a \} \}.$$

In particular, $\mathcal{C}^1[G]$ can be computed from G in polynomial time.

This just follows from the definitions, as ROMEIO can choose any string from R_a .

We next describe how each $\mathcal{C}^{k+1}[G]$ can be computed from $\mathcal{C}^k[G]$. The algorithm uses alternating nested word automata (ANWAs) which we will now define.

An *alternating nested word automaton* (ANWA) $A = (Q, \Sigma, \delta, q_0, F)$ is defined like an NWA, except that the two parts of δ map $(Q \times \langle \Sigma \rangle)$ into $\mathcal{B}^+(Q \times Q)$ and $(Q \times Q \times \langle / \Sigma \rangle)$ into $\mathcal{B}^+(Q)$, respectively, where $\mathcal{B}^+(Q)$ denotes the set of all positive boolean combinations over elements of Q using the binary operators \wedge and \vee (and likewise for $\mathcal{B}^+(Q \times Q)$).

The semantics of ANWA is defined via *runs*, which require the notion of *tree domains*. A tree domain is a prefix-closed language $D \subseteq \mathbb{N}^*$ of words over \mathbb{N} such that, if $wk \in D$ for some $w \in D$, $k \in \mathbb{N}$, then also $wj \in D$ for all $j < k$. Strings in a tree domain are interpreted as node addresses for ordered trees in the standard way: ϵ addresses the root, and if $w \in D$ addresses some node v with k children, then $w1, \dots, wk \in D$ address those children.

For any function $\lambda : D \rightarrow (Q \cup (Q \times Q))$ and node address $x \in D$, we denote by $\overline{\lambda(x)}$ the linear state component of $\lambda(x)$, i.e. if $\lambda(x) = q$ or $\lambda(x) = (q, p)$ for some $p, q \in Q$, then $\overline{\lambda(x)} = q$.

A *run* $r = (D, \lambda)$ of an ANWA A over a nested word $w = w_1 \dots w_n$ is a finite tree of depth n , represented by a tree domain D and a labelling function $\lambda : D \rightarrow (Q \cup (Q \times Q))$ such that $\lambda(\epsilon) = q_0$ and, for every $x \in D$ of length i with ℓ children, it holds that

- if $w_{i+1} \in \langle \Sigma \rangle$, then $\{ \lambda(x \cdot 1), \dots, \lambda(x \cdot \ell) \} \models \delta(\overline{\lambda(x)}, w_{i+1})$, and
- if $w_{i+1} \in \langle / \Sigma \rangle$ with associated opening tag w_j , and $\lambda(y) = (q, p)$ for some $p, q \in Q$ (where y is the prefix of x of length j), then $\{ \lambda(x \cdot 1), \dots, \lambda(x \cdot \ell) \} \models \delta(\lambda(x), p, w_{i+1})$.

An ANWA A *accepts* a nested word w if there is a run (D, λ) over w such that $\lambda(x) \in F$, for every $x \in D$ of length $|w|$.

ANWAs are used twice in the generic algorithm, first, to inductively compute $\mathcal{C}^{k+1}[G]$ from $\mathcal{C}^k[G]$, second to actually decide $\text{JWin}(G)$, given $\mathcal{C}[G]$. The following proposition will be crucial, in both cases.

Proposition 6. There is an algorithm that computes from the call effect $\mathcal{C}[G]$ of a game G in polynomial time in $|\mathcal{C}[G]|$ and $|G|$ an ANWA $A_{\mathcal{C}[G]}$ such that $L(A_{\mathcal{C}[G]}) = \text{JWin}(G)$.

The computation of $\mathcal{C}^{k+1}[G]$ from $\mathcal{C}^k[G]$ involves a non-emptiness test for ANWAs, the second stage a test whether $w \in L(A_{\mathcal{C}[G]})$. Therefore, both of the following complexity results for ANWAs influence the complexity of our algorithms.

Proposition 7.

- (a) Non-emptiness for ANWAs is 2-EXPTIME-complete.
- (b) The membership problem for ANWAs is PSPACE-complete.

Statement (a) follows immediately from the corresponding result for visibly pushdown automata in [5], statement (b) is new, to the best of our knowledge, and seems to be interesting in its own right. It is shown in the appendix.

Now we continue describing the ingredients of the first stage of the generic algorithm.

Lemma 8. Given a state $q \in Q$, an alphabet symbol $a \in \Gamma$, and $\mathcal{C}^k[G]$, for some $k \geq 1$, the call effect $\mathcal{C}^{k+1}[G](a, q)$ can be computed in doubly exponential time in $|G|$.

By Lemmas 5 and 8, one can compute $\mathcal{C}^k[G]$ inductively, for every $k \geq 1$. By definition it holds, for every q and a , that $\mathcal{C}^k[G](a, q)$ is contained in the closure of $\mathcal{C}^{k+1}[G](q, a)$ under supersets. As there are $\leq 2^{|Q|}$ sets in each $\mathcal{C}^k[G](a, q)$ (for $a \in \Gamma, q \in Q$), the computation reaches a fixed point after at most exponentially many iterations. We denote this fixed point by $\mathcal{C}^*[G]$, that is, we define, for every $a \in \Sigma, q \in Q$:

$$\mathcal{C}^*[G](a, q) \stackrel{\text{def}}{=} \left[\bigcup_{k=1}^{\infty} \mathcal{C}^k[G](a, q) \right]_{\min}$$

In particular, for each game G , there is a number $\ell \leq |\Gamma| \times |Q| \times 2^{|Q|}$ such that $\mathcal{C}^*[G] = \mathcal{C}^\ell[G]$ and $\mathcal{C}^m[G] = \mathcal{C}^\ell[G]$, for every $m \geq \ell$. However, it is not self evident that this process actually constructs $\mathcal{C}[G]$, i.e., that $\mathcal{C}^*[G] = \mathcal{C}[G]$. The following result shows that this is actually the case.

Proposition 9. For every cfG G it holds: $\mathcal{C}^*[G] = \mathcal{C}[G]$.

Now we can give a (high-level) proof for Theorem 3.

Proof of Theorem 3. We first justify the upper bounds. Let G be a cfG and w a word. By Lemma 5, $\mathcal{C}^1[G]$ can be computed in polynomial time from G . For the replay-free case, we can immediately construct an ANWA for $\text{JWin}(G)$ and evaluate it on w , yielding a PSPACE upper bound by Proposition 7.

For (a) and (b), $\mathcal{C}[G]$ ($\mathcal{C}^k[G]$, respectively) can be computed in doubly exponential time, $A_{\mathcal{C}}$ can be computed in exponential time (in the size of G), and whether $w \in L(A_{\mathcal{C}})$ can then be tested in polynomial space in $|A_{\mathcal{C}}|$ and $|w|$, that is, in at most exponential space in $|G|$ and $|w|$.

That these upper bounds can not be considerably improved, is stated in the following proposition, thereby completing the proof of Theorem 3. \square

Proposition 10. For the class of unrestricted games JWIN is

- (a) 2-EXPTIME-hard with bounded replay, and
- (b) PSPACE-hard with no replay.

Claims (a) and (b) of Proposition 10 follow from the corresponding parts of Proposition 7; in the proof, we construct from an ANWA A a replay-free cfG simulating A on any input word w (yielding claim (b)) and explain how replay can be added to that game to find and verify a witness for the non-emptiness of A , if one exists (yielding claim (a)).

For finite (and explicitly given) replacement languages the complexity changes considerably in the cases with replay, but not in the replay-free case.

Proposition 11. For the class of unrestricted games with finite replacement languages, $\text{JWIN}(\mathcal{G})$ is

- (a) EXPTIME-complete with unbounded replay, and
- (b) PSPACE-complete with bounded or without replay.

The upper bound in (a) follows as for finite replacement languages $\mathcal{C}^{k+1}[G](a, q)$ can be computed from $\mathcal{C}^k[G](a, q)$ in polynomial space¹⁰. The PSPACE upper bound in (b) can then be achieved by the usual “recomputation technique” of space-bounded computations.

The lower bound in (a) already holds for flat words (see Theorem 4.3 in [14]). The lower bound in (b) follows as the proof of Proposition 10 only uses finite replacement languages.

As our algorithms generally construct ANWAs deciding $\text{JWin}(G)$, the data complexity for JWIN is in PSPACE for all cases considered in this section due to Proposition 7.

4 Games with XML Schema target languages

The results of Section 3 provide a solid foundation for our further studies, but the setting studied there suffers from two problems: (1) the complexities are far too high (at least for games with replay) and (2) the assumption that target and replacement languages are specified by (D)NWAs is not very realistic. In this section, we address both issues at the same time: when we require that target languages are specified by typical XML schema languages (DTD or XML Schema), we get considerably better complexities.

The better complexities basically all have the same reason: XML Schema target languages can be described by a restriction of nested word automata, which we call *simple* below. This restriction translates to the alternating NWAs corresponding to call effects. For simple ANWAs, however, the two basic algorithmic problems, Non-emptiness and Membership have dramatically better complexities: PSPACE and PTIME as opposed to 2-EXPTIME and PSPACE, respectively. We emphasise that, in accordance with the official standards, our definitions for DTDs and XML Schema require *deterministic* regular expressions.

Altogether, we prove the following complexity results.

¹⁰It is worth noting that this upper bound even holds if the finite replacement language is not explicitly given, but represented by NWAs.

Theorem 12. For classes of games with XML Schemas or DTDs, respectively, JWIN is

- (a) EXPTIME-complete for unbounded replay,
- (b) PSPACE-complete for bounded replay, and
- (c) PTIME-complete (under logspace-reductions) without replay.

Here, the lower bounds are proven for DTDs, and the upper bounds for XML Schemas.

The lower bound in Theorem 12 (b) for the case of games with DTDs contradicts the statement of a PTIME algorithm in Section 4.3 of [12] (unless $\text{PTIME} = \text{PSPACE}$).¹¹

Before we describe the proof of Theorem 12, we first define *single-type tree grammars* and *local tree grammars* as well-established abstractions of XML Schema and DTDs, respectively (see, e.g., [13]). However, we will refer to grammars of these types as XML Schemas and DTDs, respectively.

Definition 13. A (*regular*) *tree grammar* is a tuple $T = (\Sigma, \Delta, S, P, \lambda)$, where

- Σ is a finite alphabet of *labels*,
- Δ is a finite alphabet of *types*,
- $S \in \Delta$ is the *root* or *starting type*,
- P is a set of *productions* of the form $X \rightarrow r_X$ mapping each type $X \in \Delta$ to a deterministic regular expression r_X over Δ , called the *content model* of X , and
- $\lambda : \Delta \rightarrow \Sigma$ is a *labelling function* assigning a label from Σ to each type in Δ .

T is *single-type* if for each $X \in \Delta$, the content model r_X contains no competing types, i.e. if r_X contains no two types $Y \neq Z$ with $\lambda(Y) = \lambda(Z)$. T is *local*, if it has exactly one type for every label.

We omit the definition of the formal semantics of regular tree grammars. The nested word language $L(T)$ described by T is just the set of linearisations of trees of the tree language that is defined in the standard way.

We next define *simple* DNWAs, a restriction of DNWAs that captures all languages specified by single-type tree grammars. In simple DNWAs, states are typed, i.e. each state has a component in some type alphabet Δ . Informally, when a simple DNWA A reads a subword $w = \langle a \rangle v \langle /a \rangle$ in state q , it determines already on reading $\langle a \rangle$ which state q' it will take after processing w , and this state will be of the same type as q . After reading $\langle a \rangle$, the linear state of A only depends on the *type* of q , not the exact state; this models the single-type restriction. After reading $\langle a \rangle$, A goes on to validate v , and if this validation fails, A enters a failure state \perp instead of q' . Thus, the state of A at a position basically only

¹¹A close inspection of the construction in the proof in [12] reveals that the automaton constructed there does not deal correctly with the alternation between the choices of ROMEO and JULIET. More precisely, the automaton allows ROMEO to let the suffix of a replacement string depend on the choices of JULIET on its prefix.

depends on its ancestor positions (in the tree view of the document) and their left siblings. The only way in which other nodes in subtrees of these nodes can influence the state is by assuming the sink state \perp . Thus, in the spirit of [11], we could call such DNWA *ancestor-sibling-based* but we prefer the term *simple* for simplicity.

Definition 14. A deterministic NWA $A(T) = (Q, \Sigma, \delta, q_0, F)$ in normal form is *simple* (SNWA) if there exist a *type alphabet* Δ and *state set* P with $Q \subseteq P \times \Delta$, a *local acceptance function* $F_{\text{loc}} : \Sigma \rightarrow \mathcal{P}(Q)$, a *target state function* $t : Q \times \Sigma \rightarrow Q$ and a *failure state* $\perp \in Q \setminus F$, such that the following conditions are satisfied for every $a \in \Sigma$:

- for every $p, p' \in P, X \in \Delta$: $\delta((p, X), \langle a \rangle) = \delta((p', X), \langle a \rangle)$;
- for every $q \in F_{\text{loc}}(a)$: $\delta(q, p, \langle /a \rangle) = t(p, a)$;
- for every $q \in Q \setminus F_{\text{loc}}(a)$: $\delta(q, p, \langle /a \rangle) = \perp$ and
- for every $q \in Q$: $\delta(\perp, \langle a \rangle) = \delta(\perp, q, \langle /a \rangle) = \perp$.
- for every $(p, X) \in Q$: $t((p, X), a) = (p', X)$ for some $p' \in P$.

A cfG is called *simple* if its target DNWA is simple.

Proposition 15. From every single-type tree grammar T , a simple DNWA A can be computed in polynomial time, such that $L(A) = L(T)$.

The following adaptation of the notion of *simplicity* to ANWAs is a bit technical. It will guarantee however that the ANWAs obtained from simple games are simple and have reasonable complexity properties.

Definition 16. An ANWA $A = (Q, \Sigma, \delta, q_0, F)$ with $Q \subseteq P \times \Delta$ (for some state set P and type alphabet Δ) is *simple* (SANWA), if it has the following two properties.

- (*Horizontal simplicity*) There are a *local acceptance function* $F_{\text{loc}} : \Sigma \rightarrow \mathcal{P}(Q)$, a *test state* $q_?$ $\in Q$, and a *target state function* $t : Q \times \Sigma \rightarrow Q$, such that the transition function δ of A satisfies the following conditions:

$$\begin{aligned} & - \delta(q, q', \langle /a \rangle) = t(q', a) \text{ for all } q \in Q \text{ and } q' \neq q_?; \\ & - \delta(q, q_?, \langle /a \rangle) = \begin{cases} \text{true,} & \text{if } q \in F_{\text{loc}}(a) \\ \text{false,} & \text{if } q \notin F_{\text{loc}}(a) \end{cases} \end{aligned}$$

Furthermore, for each $(p, X) \in Q$ and $a \in \Sigma$, it holds that $t((p, X), a) = (p', X)$ for some $p' \in P$.

- (*Vertical Simplicity*) For each $X \in \Delta$ and $a \in \Sigma$, there is a $q \in Q$ such that for all $p \in P$ it holds that $\delta((p, X), \langle a \rangle) \in \mathcal{B}^+(\{q\} \times ((P \times \{X\}) \cup \{q_?\}))$.

Essentially, horizontal simplicity states that A has two kinds of computations on a well-nested subword: (1) computations starting from a pair $(q, q_?)$ test a property of the subword and can either succeed or fail at the end of the subword (and thus influence the overall computation); (2) computations starting from a pair (q, q') for $q' \neq q_?$ basically ignore the subword. Even though they may

branch in an alternating fashion, the state after the closing tag $\langle /a \rangle$ is the same in all subruns, is determined by $t(q', a)$ and has the same type as q' .

Vertical simplicity, on the other hand, states that all alternation in A happens in the choice of hierarchical states – while, on an opening tag, A may branch into sub-runs pushing different hierarchical states onto the stack, the choice of linear follow-up state is “locally deterministic”, depending only the type of the previous state of A and the label of the tag being read, and the current type is preserved in all hierarchical states except for $q_?$. Together, these two conditions also guarantee that SNWAs may also be interpreted as SANWAs.

Proposition 17.

- (a) Non-emptiness for SANWA is PSPACE-complete.
- (b) The membership problem for SANWA is decidable in polynomial time.

Proof of Theorem 12. The generic algorithm from the previous section can be adapted for simple cfGs, but with better complexity thanks to Proposition 17, to yield the upper bounds stated in Theorem 12.

More precisely, Proposition 17 (b) and Lemma 5 yield a polynomial time bound for replay-free games. Proposition 17 (a) guarantees that the inductive step in the computation of $\mathcal{C}[G]$ can be carried out in polynomial space (as opposed to doubly exponential time).¹² The upper bounds for games with unrestricted replay follows immediately and the upper bound for bounded replay can be shown similarly as in Proposition 11 (b).

The lower bounds are given by the following proposition. They mostly follow from careful adaptation of lower bound proofs of [14] for games on flat strings. \square

Proposition 18. For the class of games with target languages specified by DTDs, JWIN is

- (a) EXPTIME-hard with unrestricted replay,
- (b) PSPACE-hard with bounded replay, and
- (c) PTIME-hard (under logspace-reductions) without replay

For finite (and explicitly given) replacement languages we get feasibility even for bounded replay, but no improvement for unbounded replay.

Proposition 19. For the class of games with target languages specified by XML Schemas and explicitly enumerated finite replacement languages, JWIN is

- (a) EXPTIME-complete with unrestricted replay, and
- (b) PTIME-complete (under logspace-reductions) with bounded replay or without replay.

The same results hold for DTDs in place of XML Schemas.

Once again, as our algorithm generally computes a SANWA deciding $\text{JWin}(G)$, the data complexity for JWIN is in PTIME for all cases considered here, due to Proposition 17.

¹²We actually use a slightly stronger result than Proposition 17 (a): deciding whether, for an NWA A_1 and a SANWA A_2 , it holds $L(A_1) \cap L(A_2) \neq \emptyset$, is complete for PSPACE.

5 Validation of parameters and Insertion

In this section, we focus on two features that have not been addressed in the previous two sections: validation of the parameters of a function call with respect to a given schema, and a semantics which allows that returned trees do not *replace* their call nodes but are inserted next to them.

5.1 Validation of parameters

As pointed out in [12], in Active XML, parameters of function calls should be valid with respect to some schema. Transferred to the setting of cfGs this means that JULIET should only be able to play a Call move in a configuration $(J, u\langle a \rangle v, \langle /a \rangle w)$ if $\langle a \rangle v \langle /a \rangle$ is in V_a for some set V_a of words that are valid for calls of $\langle /a \rangle$. Our definition of cfGs and the previous ones studied in the literature mostly ignore this aspect.¹³ We do not investigate all possible game types in combination with parameter validation but rather concentrate on the most promising setting with respect to tractable algorithms. It turns out, that games without replay and with DTDs to specify target, replacement and validation languages have a tractable winning problem as long as the number of different validation DTDs is bounded by some constant.¹⁴ It becomes intractable if the number of validation schemas can be unbounded and (already) with target and validation languages specified by XML Schemas, even with only one validation schema.

More precisely, we prove the following results.

Theorem 20. For the class of games with validation with a bounded number of validation DTDs and target languages specified by DTDs, JWIN is in PTIME without replay.

The algorithm uses a bottom-up approach. The basic idea is that, starting from the leaves, at each level of the tree (that is for some node v and its leaf children) all relevant information about the game in the subtree t_v is computed with the help of flat replay-free games and aggregated in v . Then the children of v are discarded and the algorithm continues until only the root remains.

The following result shows that for slightly stronger games, parameter validation worsens the complexity.¹⁵

Theorem 21. For the class of games with validation, JWIN (without replay) is

- (a) EXPTIME-hard, if target and validation languages are specified by DNWAs (even with only one function symbol);
- (b) PSPACE-hard, for games with only one function symbol, if the validation language is given by an XML schema, the target language by a DTD and a finite replacement language; and

¹³Actually, [12] takes validation into account but the precise way in which parameters are specified and tested is not explained in full detail.

¹⁴Note that this implies a polynomial-time data complexity for arbitrary replay-free games with DTD target, replacement and validation languages.

¹⁵This is, of course not surprising. If any, the surprising result is Theorem 20.

- (c) PSPACE-hard, for games with an unbounded number of validation DTDs and replacement and target languages specified by DTDs.

Part (a) is proven by reduction from the intersection emptiness problem for DNWAs, while parts (b) and (c) use similar reductions from the problem of determining whether a quantified Boolean formula in disjunctive normal form is true.

Due to time constraints and as we are mainly interested in finding tractable cases, we have not looked for matching upper bounds.

5.2 Insertion rules

In our definition of Call moves, we define the successor configuration of a configuration $(R, u\langle a \rangle v, \langle /a \rangle w)$ to be $(J, u, v'w)$, that is, $\langle a \rangle v \langle /a \rangle$ is *replaced* by a string $v' \in R_a$. However, Active XML also offers an “append” option, where results of function calls are inserted as siblings after the calling function node (cf. [1]). There are (at least) three possible semantics of a Call move for insertion (as opposed to replacement) based games: the next configuration could be (1) $(J, u, \langle a \rangle v \langle /a \rangle v'w)$, (2) $(J, u\langle a \rangle v \langle /a \rangle, v'w)$, or (3) $(J, u\langle a \rangle v \langle /a \rangle v', w)$, depending on “how much replay” we allow for JULIET. We consider (1) as the general setting, (2) as the setting with *weak replay* and (3) as the setting *without replay*. It turns out that the weak replay setting basically corresponds to the (unrestricted) setting with replacement rules and that (3) corresponds to the replay-free setting with replacement rules. Setting (1), however, gives JULIET a lot of power and makes $\text{JWIN}(\mathcal{G})$ undecidable.

Theorem 22. For the class of games with insertion semantics, target DNWAs and replacement NWAs, JWIN is

- (a) undecidable in general;
- (b) 2-EXPTIME-complete for games with weak replay; and
- (c) PSPACE-complete for games without replay.

The proof idea for Theorem 22 is to simulate insertion-based games by replacement-based games and vice versa; part (a) additionally uses the undecidability of JWIN for arbitrary (i.e. not necessarily left-to-right) strategies on games with flat strings, which was proven to be undecidable in [14].

6 Conclusion

The complexity of context-free games on nested words differs considerably from that on flat words (2-EXPTIME vs. EXPTIME), but there are still interesting tractable cases. One of the main insights of this paper is that the main tractable cases remain tractable if one allows XML Schema instead of DTDs for the specification of schemas.

Another result is that adding validation of input parameters can worsen the complexity, but tractability can be maintained by a careful choice of the setting. However, here the step from DTDs to XML Schema may considerably worsen the complexity.

Insertion semantics with unlimited replay yields undecidability.

We leave open some corresponding upper bounds in the setting with validation of input parameters. In future work, we plan to study the impact of parameters of function calls more thoroughly.

A Appendix

For easier reference, we restate the results that were already stated in the body of the paper. Definitions and results not stated in the body can be identified by their number of the type A.xxx. At the end of the appendix there is another bibliography which contains references for all work mentioned in the appendix.

Proofs for Section 2

Lemma 2 (*restated*). *There is a polynomial-time algorithm that computes for every deterministic NWA an equivalent deterministic NWA in normal form.*

Proof. Let $A = (Q, \Sigma, \delta, q_0, F)$ and let δ_1 and δ_2 the projections of δ to its first and second component (for opening tags only), respectively, i.e., $\delta(p, \langle a \rangle) = (\delta_1(p, \langle a \rangle), \delta_2(p, \langle a \rangle))$. An equivalent DNWA $A' = (Q, \Sigma, \delta', q_0, F)$ in normal form can be constructed by letting $\delta'(p, \langle a \rangle) \stackrel{\text{def}}{=} (\delta_1(p, \langle a \rangle), p)$ and $\delta'(q, p, \langle /a \rangle) \stackrel{\text{def}}{=} \delta(q, \delta_2(p, \langle a \rangle), \langle /a \rangle)$. \square

Proofs for Section 3

In this section, we give proofs for the upper and lower bounds on the complexity of JWIN for unrestricted games stated in Section 3.

Upper bounds for Theorem 3

The proof of the upper bounds in Theorem 3 consists technically of three main parts:

- the first part describes how to compute an ANWA for a cfG from its call effect (Proposition 6),
- the second part establishes the complexity of emptiness and membership for ANWAS (Proposition 7), and
- the third part shows that the fix point process sketched after Lemma 8 in Section 3 indeed computes the call effect of a game.

Transforming call effects into ANWAs

The proof of Proposition 6 requires a considerable amount of preparation.

As mentioned in Section 3, our main tool for proving upper bounds on general cfGs is abstracting from subgames to the effects they induce on the target automaton $A(T)$. To facilitate the proof of Proposition 6, we extend this abstraction from the *call effects* of subgames on rooted strings as defined in Section 3 to effects of arbitrary nested strings. Formally, a *(word) effect* maps states q of $A(T)$ to sets of sets of states of $A(T)$. The effect of a game G on a word w relative to state q is basically the set of all state sets X , for which JULIET has a strategy that guarantees that every play on w yields some word v with $\delta^*(q, v) \in X$. For ease of reference, we restate some definitions from Section 2 needed for word effects.

In the following, we sometimes consider *subgames* on a certain part of a string and talk about strategies for subgames. From a configuration (u, vw) , JULIET can use a strategy σ on the subgame on v . This means that she follows σ until a configuration (uv', w) is reached.

Definition A.1. For a cfG $G = (\Sigma, \Gamma, R, T)$ with a deterministic target NWA $A(T) = (Q, \Sigma, \delta, q_0, F)$, we define the following notation.

- $\text{word}_G(w, \sigma, \tau)$ denotes the unique final word that is reached in the game on w with strategies $\sigma \in \text{STRAT}_J(G)$ and $\tau \in \text{STRAT}_R(G)$.
- $\text{words}_G(w, \sigma) \stackrel{\text{def}}{=} \{\text{word}_G(w, \sigma, \tau) \mid \tau \in \text{STRAT}_R\}$ denotes the set of final words that can be reached through strategies of ROMEO, for a fixed strategy $\sigma \in \text{STRAT}_J(G)$.
- $\text{states}_G(q, w, \sigma) \stackrel{\text{def}}{=} \{\delta^*(q, v) \mid v \in \text{words}_G(w, \sigma)\}$ denotes the set of states that $A(T)$ can take at the end of final words that can be reached through strategies of ROMEO, for a fixed strategy $\sigma \in \text{STRAT}_J(G)$.

Finally, we define the *word effect*, $\mathcal{E}[G, w] : Q \rightarrow \mathcal{P}(\mathcal{P}(Q))$, of G on w by

$$\mathcal{E}[G, w](q) \stackrel{\text{def}}{=} [\{\text{states}_G(q, w, \sigma) \mid \sigma \in \text{STRAT}_J(G)\}]_{\min},$$

for every $q \in Q$, where the operator $[\cdot]_{\min}$ removes all non-minimal sets from a set of sets as before.

To simplify notation, the subscript G will often be omitted if the game G is clear from the context.

The intuition behind word effects is the following abstraction of cfGs into single-round games: On an input string w , JULIET first chooses a strategy σ , then ROMEO chooses a strategy τ ; the outcome of the game on w is uniquely determined by σ and τ . In terms of effects, this corresponds to JULIET picking a set $X = \text{states}_G(q_0, w, \sigma) \in \mathcal{E}[G, w](q_0)$ and ROMEO then choosing a final state $q = \delta^*(q_0, \text{word}_G(w, \sigma, \tau)) \in X$. This intuition also explains our use of the $[\cdot]_{\min}$ operator, as it makes no sense for JULIET to offer ROMEO a choice from a set $X \subseteq Q$ if she can instead offer him the more limited options in some $X' \subsetneq X$.¹⁶

It is easy to see that JULIET has a winning strategy in G on w if and only if there is some $X \in \mathcal{E}[G, w](q_0)$ such that $X \subseteq F$; to determine whether JULIET has a winning strategy it therefore suffices to compute $\mathcal{E}[G, w]$.

It is natural to reason about effects for nested words in an inductive fashion. We first consider sequential composition. From JULIET's point of view, the game on a nested word uv (with $u, v \in \text{WF}(\Sigma)$) from a state q on proceeds as follows. JULIET fixes a strategy σ on u . The set of states that ROMEO can reach at the end of the subgame on u is just $\text{states}_G(q, u, \sigma)$. For each state $p \in \text{states}_G(q, u, \sigma)$, JULIET can choose a strategy σ_p for v and the result set is then the union of all sets that can be reached by ROMEO against any σ_p on v . To express the set of all combinations of outcomes for the second part, we use the following operator.

Definition A.2. Let $\mathcal{D} = \{D_1, \dots, D_n\}$ be a set of sets of sets. Then $\text{MIX}(\mathcal{D})$ is the set

$$\{[d_1 \cup \dots \cup d_n \mid d_1 \in D_1 \wedge \dots \wedge d_n \in D_n]\}_{\min}.$$

In other words, the MIX operation yields every way of taking the union of one element from each of D_1, \dots, D_n and then removes non-minimal sets.

Let E_1, E_2 be mappings from Q into $\mathcal{P}(\mathcal{P}(Q))$. Then the *composition* of E_1 and E_2 is defined as the mapping $E_1 \circ E_2 : Q \rightarrow \mathcal{P}(\mathcal{P}(Q))$ with

$$(E_1 \circ E_2)(q) \stackrel{\text{def}}{=} \left[\bigcup_{X \in E_1(q)} \text{MIX}(\{E_2(q') \mid q' \in X\}) \right]_{\min}.$$

Not surprisingly, effect composition commutes with word concatenation.

Lemma A.3. For every cfG $G = (\Sigma, \Gamma, R, T)$ and $u, v \in \text{WF}(\Sigma)$ it holds

$$\mathcal{E}[G, uv] = \mathcal{E}[G, u] \circ \mathcal{E}[G, v].$$

Before proving Lemma A.3, we give an auxiliary result that will greatly simplify proofs about effects and similar functions. To that end, we call a set \mathcal{D} of sets *normalised* if it contains no two sets X, Y such that $X \subsetneq Y$ (or, equivalently, if $\mathcal{D} = [\mathcal{D}]_{\min}$). For two sets of sets E_1, E_2 , we write $E_1 \supseteq E_2$ if and only if every $X \in E_1$ has a subset in E_2 .

¹⁶Minimisation in our model corresponds to the monotonicity of powers [17] or effectivity functions [15]. Using, as we do, an inclusion-minimal “basis” instead of a monotonic “upward closure” allows for a more succinct representation and lower complexity in some places.

Lemma A.4. Let E_1, E_2 be two normalised sets of sets. If $E_1 \supseteq E_2$ and $E_1 \subseteq E_2$, then $E_1 = E_2$.

Proof. We prove only $E_1 \subseteq E_2$; inclusion in the other direction then follows by symmetry. Let $X_1 \in E_1$, and let $X_2 \in E_2$ with $X_2 \subseteq X_1$. By assumption, there also exists $X'_1 \in E_1$ with $X'_1 \subseteq X_2$, and therefore $X'_1 \subseteq X_2 \subseteq X_1$. Since both X_1 and X'_1 are in E_1 , and E_1 is normalised by assumption, this inclusion cannot be proper, and it follows that $X'_1 = X_2 = X_1$, and therefore $X_1 = X_2$ and $X_1 \in E_2$. \square

Proof of Lemma A.3. Let $q \in Q$. This proof uses Lemma A.4 to prove the equality of the two normalised sets $\mathcal{E}[G, uv](q)$ and $(\mathcal{E}[G, u] \circ \mathcal{E}[G, v])(q)$.

(\supseteq): Let $X \in \mathcal{E}[G, uv](q)$. Then there exists some strategy $\sigma_{uv} \in \text{STRAT}(G)$ such that $X = \text{states}_G(q, uv, \sigma_{uv})$. Let σ_u be the restriction of σ_{uv} to the subgame on u , let $X_u \in \mathcal{E}[G, u](q)$ with $X_u \subseteq \text{states}_G(q, u, \sigma_u)$ and $\{p_1, \dots, p_k\} = X_u$. For each $i \in [k]$, let σ_v^i be a restriction of σ_{uv} to the subgame on v in case ROMEO chooses a strategy τ with $\text{state}_G(q, u, \sigma, \tau) = p_i$, and let $X_v^i \in \mathcal{E}[G, v](p_i)$ such that $X_v^i \subseteq \text{states}_G(p_i, v, \sigma_v^i)$ for all $i \in [k]$. Let $X' = X_v^1 \cup \dots \cup X_v^k$.

By definition of \circ , and because of normalisation, there exists some $X'' \in (\mathcal{E}[G, u] \circ \mathcal{E}[G, v])(q)$ with $X'' \subseteq X'$. So, to show the desired inclusion, it suffices to prove that $X' \subseteq X$.

Let $p' \in X'$. Then, $p' \in X_v^i$ and therefore $X' \in \text{states}_G(p_i, v, \sigma_v^i)$ for some $i \in [k]$. Also, $p_i \in X_u \subseteq \text{states}_G(q, u, \sigma_u)$, i.e. $p_i = \text{state}_G(q, u, \sigma_u, \tau)$ for some $\tau \in \text{STRAT}_R(G)$. By the definition of σ_u and σ_v^i , this implies that $p' \in \text{states}_G(q, uv, \sigma_{uv}) = X$.

(\subseteq): Let $X \in (\mathcal{E}[G, u] \circ \mathcal{E}[G, v])(q)$. By definition of \circ , there are sets $X_u = \{q_1, \dots, q_k\} \in \mathcal{E}[G, u](q)$ and $X_v^i \in \mathcal{E}[G, v](q_i)$ for each $i \in [k]$ such that $X = X_v^1 \cup \dots \cup X_v^k$. By definition of $\mathcal{E}[G, \cdot]$, there are strategies $\sigma_u, \sigma_v^1, \dots, \sigma_v^k \in \text{STRAT}(G)$ with $X_u = \text{states}_G(q, u, \sigma_u)$ and $X_v^i = \text{states}_G(q_i, v, \sigma_v^i)$.

Define a strategy σ_{uv} on uv as follows. On u , JULIET plays according to σ_u ; if this play yields some string $u_i \in \text{words}_G(u, \sigma_u)$ with $\delta^*(q, u_i) = q_i$, JULIET then plays according to σ_v^i on v .

Denote $\text{states}_G(q, uv, \sigma_{uv})$ by X' for short. Due to normalisation, there exists some $X'' \in \mathcal{E}[G, uv]$ with $X'' \subseteq X'$. What needs to be shown is therefore only that $X' \subseteq X$.

Let $q' \in X'$. Then there exists a strategy τ for ROMEO and strings $u', v' \in \text{WF}(\Sigma)$ such that $u'v' = \text{word}_G(uv, \sigma_{uv}, \tau)$, $u' = \text{word}_G(u, \sigma_{uv}, \tau)$ and $\delta^*(q, u'v') = q'$. Let $q_u = \delta^*(q, u')$; then, it holds that $q' = \delta^*(q_u, v')$. By the definition of σ_{uv} , it follows that $q_u \in X_u$ and $q' \in X_v^i$ for some i , so $q' \in X$, which concludes the proof. \square

It follows directly from Lemma A.3 that the sequential composition of effects is associative.

The word effect of a word of the form $\langle a \rangle v \langle /a \rangle$ is induced by the word effect of v and the possible moves of the players on $\langle a \rangle$ and $\langle /a \rangle$. In particular, as JULIET may choose Call on $\langle /a \rangle$, the possible outcomes of a subgame on a subword of the form $\langle a \rangle \langle /a \rangle$ become crucial. As in the main part of this paper, we summarise the possible outcomes of subgames on “two-letter words” of the form $\langle a \rangle \langle /a \rangle$ by *call effects* as defined in Definition 4. For ease of reference, we restate that

$$\mathcal{C}[G](a, q) \stackrel{\text{def}}{=} [\{\text{states}_G(q, \langle a \rangle \langle /a \rangle, \sigma) \mid \sigma \in \text{STRAT}_{J, \text{Call}}(G)\}]_{\min},$$

for every $a \in \Sigma$ and $q \in Q$, where $\text{STRAT}_{\text{J,Call}}(G)$ contains all strategies of JULIET that start by playing Read on $\langle a \rangle$ and Call on $\langle /a \rangle$.

To describe hierarchical composition of word effects we define, for every $a \in \Sigma$ the following operator $H_a : Q \rightarrow \mathcal{P}(\mathcal{P}(Q))$. For every two functions $E : Q \rightarrow \mathcal{P}(\mathcal{P}(Q))$ and $C : \Sigma \times Q \rightarrow \mathcal{P}(\mathcal{P}(Q))$ and $q, q' \in Q$ such that $\delta(q, \langle a \rangle) = q'$, let

$$H_a[E, C](q) \stackrel{\text{def}}{=} \left[\bigcup_{X \in E(q')} \text{MIX}(\{C(a, q) \cup \{\delta(r, q, \langle /a \rangle)\} \mid r \in X\}) \right]_{\min}.$$

Informally, interpreting E as a word effect and C as a call effect, the first set inside the MIX operator accounts for Call moves and the second for Read moves of JULIET. Now we can formulate how effects behave hierarchically.

Lemma A.5. For every cfG $G = (\Sigma, \Gamma, R, T)$, $v \in \text{WF}(\Sigma)$, and $a \in \Sigma$, it holds

$$\mathcal{E}[G, \langle a \rangle v \langle /a \rangle] = H_a[\mathcal{E}[G, v], \mathcal{C}[G]].$$

Proof. We show, once again using Lemma A.4, that for every $q \in Q$, it holds that

$$\mathcal{E}[G, \langle a \rangle v \langle /a \rangle](q) = H_a[\mathcal{E}[G, v], \mathcal{C}[G]](q).$$

(\supseteq): Let $X \in \mathcal{E}[G, \langle a \rangle v \langle /a \rangle](q)$. Then there is some strategy $\sigma \in \text{STRAT}(G)$ such that $X = \text{states}_G(q, \langle a \rangle v \langle /a \rangle, \sigma)$. Let σ_v be the sub-strategy of σ on v , let $\delta(q, \langle a \rangle) = (q_a, p)$ and let $X_v = \{q_1, \dots, q_k\} = \text{states}_G(q_a, v, \sigma_v)$. For each $i \in [k]$, let $v_i \in \text{words}_G(v, \sigma_v)$ such that $\delta^*(q_a, v_i) = q_i$ and let σ_i be the sub-strategy of σ starting at $(\langle a \rangle v_i, \langle /a \rangle)$. If JULIET's move on $\langle /a \rangle$ according to σ_i is Read, let $X_i = \{\delta(q_i, p, \langle /a \rangle)\}$, otherwise let $X_i = \text{states}_G(q, \langle a \rangle \langle /a \rangle, \sigma_i)$ ¹⁷. Clearly, $X' = X_1 \cup \dots \cup X_k$ has a subset in $\text{MIX}(\{\mathcal{C}[G](a, q) \cup \{\delta(r, p, \langle /a \rangle)\} \mid r \in X\})$ and therefore in $H_a[\mathcal{E}[v], \mathcal{C}[G]](q)$. It remains to be proven that $X' \subseteq X$.

Let $q' \in X'$. Then $q' \in X_i$ for some $i \in [k]$. If $X_i = \{\delta(q_i, p, \langle /a \rangle)\}$, then clearly $q' = \delta^*(q, \langle a \rangle v_i \langle /a \rangle) \in \text{states}_G(q, \langle a \rangle v \langle /a \rangle, \sigma) = X$. Otherwise, $q' \in \text{states}_G(q, \langle a \rangle \langle /a \rangle, \sigma_i)$ and σ_i coincides on $\langle a \rangle \langle /a \rangle$ with σ on $\langle a \rangle v_i \langle /a \rangle$, it follows again that $q' \in X$.

(\subseteq): Let $X \in H_a[\mathcal{E}[G, v], \mathcal{C}[G]](q)$ and let $\delta(q, \langle a \rangle) = (q', p)$. Then, there exists some $X_v = \{q_1, \dots, q_k\} \in \mathcal{E}[G, v](q')$ such that $X = X_1 \cup \dots \cup X_k$, where each X_i is either in $\mathcal{C}[G](a, q)$ or of the form $\{\delta(q_i, p, \langle /a \rangle)\}$. By the definition of $\mathcal{E}[G, v]$, there exists some strategy $\sigma_v \in \text{STRAT}$ on v such that $\text{states}_G(q', v, \sigma_v) = X_v$, and by the definition of $\mathcal{C}[G](a, q)$, for each i with $X_i \in \mathcal{C}[G](a, q)$ there exists a strategy $\sigma_i \in \text{STRAT}_{\text{J,Call}}$ such that $X_i = \text{states}_G(q, \langle a \rangle \langle /a \rangle, \sigma_i)$. We extend σ_v to a strategy σ on $\langle a \rangle v \langle /a \rangle$ as follows: JULIET reads the initial $\langle a \rangle$, then plays on v according to σ_v . The string v' resulting from this play on v has to fulfil $\delta^*(q', v') = q_i$ for some $i \in [k]$; if, for this i it holds that $X_i = \{\delta(q_i, p, \langle /a \rangle)\}$, then JULIET plays Read on $\langle /a \rangle$, otherwise she plays Call on $\langle /a \rangle$ and plays according to σ_i in the resulting sub-game. Let $X' = \text{states}_G(q, \langle a \rangle v \langle /a \rangle, \sigma)$; it is easy to see that $X' \subseteq X$, and since X' has a subset in $\mathcal{E}[G, \langle a \rangle v \langle /a \rangle](q)$ by definition, this proves the claim. \square

We are now ready to define the ANWA A_C from Proposition 6. The intuition behind it is that A_C uses alternation to guess strategy choices for JULIET and

¹⁷As, in this case, $\sigma_i \in \text{STRAT}_{\text{J,Call}}$ is a strategy playing Call on $\langle /a \rangle$, we can omit v here.

ROMEO in the above abstraction of G on w using call effects and tracks a current state q in the target language DNWA $A(T)$. On opening tags, as well as on closing tags for which A_C existentially guesses JULIET's move to be Read, A_C simply simulates $A(T)$; on closing tags $\langle /a \rangle$ where A_C decides for JULIET to play Call, A_C then chooses existentially a set $X \in \mathcal{C}[G](a, q)$ (corresponding to a substrategy for JULIET after the Call on $\langle /a \rangle$) and branches universally into all states $q' \in X$ (corresponding to ROMEO's choice of a counter-strategy and a corresponding resulting state).

Formally, $A_C = (Q, \Sigma, \delta_C, q_0, F)$ is an ANWA in normal form, where δ_C is defined as follows. (Recall that $A(T) = (Q, \Sigma, \delta, q_0, F)$ is the target language DNWA in normal form.)

- For $a \in \Sigma, q \in Q$:

$$\delta_C(q, \langle a \rangle) \stackrel{\text{def}}{=} \delta(q, \langle a \rangle).$$

- For $a \in \Sigma, q, p \in Q$:

$$\delta_C(q, p, \langle /a \rangle) \stackrel{\text{def}}{=} \delta(q, p, \langle /a \rangle) \vee \bigvee_{X \in \mathcal{C}[G](a, p)} \bigwedge_{r \in X} r.$$

We go on to prove the correctness of A_C . To that end, we call a run ρ of an ANWA A on a string w *minimal* if no proper subtree of ρ is a run of A on w (i.e. if each set of states chosen to follow up some state on reading some symbol is inclusion-minimal among the sets of states fulfilling the corresponding transition formula).

Lemma A.6. Let $q \in Q$, $w \in \text{WF}(\Sigma)$ and $X \subseteq Q$. Then, $X \in \mathcal{E}[G, w](q)$ if and only if there is a minimal run of A_C on w starting at q and ending in states from X .

Proof. Let $q \in Q$, $X \subseteq Q$ and $w \in \text{WF}(\Sigma)$. The proof is by induction on the structure of w .

For $w = \epsilon$, the claim is trivially fulfilled, as $\mathcal{E}[G, \epsilon](q) = \{\{q\}\}$ by the definition of string effects.

Let $w = uv$ for $u, v \in \text{WF}(\Sigma)$. For the “only if” direction, it follows from Lemma A.3 that there are sets $X_u = \{q_i, \dots, q_k\} \in \mathcal{E}[G, u](q)$ and X_v^1, \dots, X_v^k with $X_v^i \in \mathcal{E}[G, v](q_i)$ for each $i \in [k]$ and $X = X_v^1 \cup \dots \cup X_v^k$. By induction, there exist a minimal run ρ_u of A_C starting at q and ending inside X_u and for each $i \in [k]$ a minimal run ρ_v^i on v starting at q_i and ending inside X_v^i . From these, we can construct a run ρ of A_C on w by replacing each leaf labelled q_i in ρ_u with the entire run ρ_v^i rooted at q_i . Obviously, ρ is a run of A_C starting at q and ending inside X , and ρ is minimal because ρ_u and all ρ_v^i are. The “if” direction is proven analogously.

Let $w = \langle a \rangle v \langle /a \rangle$ for $a \in \Sigma$, $w \in \text{WF}(\Sigma)$. Let further $\delta(q, \langle a \rangle) = q'$. For “only if”, Lemma A.5 implies that $X \in H_a[\mathcal{E}[G, v], \mathcal{C}[G]](q)$. This means that there is a set $X_v = \{q_1, \dots, q_k\} \in \mathcal{E}[G, v](q')$ and sets X_w^1, \dots, X_w^k such that $X = X_w^1 \cup \dots \cup X_w^k$ and for each $i \in [k]$ either $X_w^i \in \mathcal{C}[G](a, p)$ or $X_w^i = \{\delta(q_i, q, \langle /a \rangle)\}$. By induction, there exists a minimal run ρ_v of A_C on v starting at q' and ending inside X_v . We extend ρ_v to a run ρ on w as follows: The root of ρ is labelled q and has as its only child the root of a copy of ρ_v ; each leaf of this copy labelled q_i has as its children exactly the states in X_w^i . Using the

definition of A_C , it is easy to verify that ρ is indeed a run of A_C on w , and it is also clear that ρ starts at q and ends inside X . Finally, ρ is minimal because its subrun on v is minimal, and for each q_i , the set X_w^i is an inclusion-minimal set fulfilling $\delta_C(q_i, q, \langle /a \rangle)$ (for $X_w^i \in \mathcal{C}[G](a, q)$, this follows from $\mathcal{C}[G](a, q)$ being normalised). Again, the “if” part is proven analogously. \square

Now we are in the position to prove Proposition 6:

Proposition 6 (restated). *There is an algorithm that computes from the call effect $\mathcal{C}[G]$ of a game G in polynomial time in $|\mathcal{C}[G]|$ and $|G|$ an ANWA A_C such that $L(A_C) = JWin(G)$.*

Proof. The statement follows from Lemma A.6, as A_C has an accepting run on any string $w \in WF(\Sigma)$ if and only if it has a minimal such run. Obviously, A_C is of polynomial size in the size of G and $\mathcal{C}[G]$ and can be constructed from these in polynomial time \square

The complexity of ANWAs

Proposition 7 (restated).

- (a) *Non-emptiness for ANWAs is 2-EXPTIME-complete.*
- (b) *The membership problem for ANWAs is PSPACE-complete.*

Proof. Statement (a) follows easily from [5] where 2-EXPTIME-completeness of Emptiness for alternating visibly pushdown automata was shown. The lower bound in that paper only requires finite well-nested words.

Towards the upper bound in (b), it is easy to see that an ANWA $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ on some nested word w can be simulated by an alternating Turing machine with polynomial time bound, hence the classical results from [6] yield a polynomial space upper bound.

For future reference we note that this computation can be actually be done in polynomial space in $|w|$ and the size $|Q|$ of \mathcal{A} ’s set of states, if it can be tested in polynomial space, whether

- for a given set $X \subseteq Q \times Q$ of pairs of states, a symbol a and a state q , whether $X \models \delta(q, a)$, and
- for a given set $X \subseteq Q$ of states, a symbol a and states p, q , whether $X \models \delta(q, p, a)$.

The proof of this statement is along the same lines as the proof that alternating polynomial time is contained in polynomial space: The tree of all possible computations has polynomial depth and can be analysed with polynomial space.

The lower bound in (b) is shown by a reduction from QBF, that is, the problem to decide whether a quantified Boolean formula evaluates to true. We assume that the input formula for QBF is of the form $\Phi = Q_1 x_1 \dots Q_n x_n \varphi(x_1, \dots, x_n)$ with $Q_i \in \{\exists, \forall\}$ and a boolean formula φ with m clauses in conjunctive normal form.

The idea behind this reduction is to transform Φ into an ANWA A and a nested string w such that Φ is true if and only A accepts w . Actually, w is of a very simple form: $\langle v_1 \rangle \dots \langle v_n \rangle \langle X \rangle \langle /X \rangle \langle /v_n \rangle \dots \langle /v_1 \rangle$.

If the automaton A reads an opening tag $\langle v_i \rangle$, it branches existentially, if x_i is existentially quantified, and it branches universally, if x_i is universally quantified, thus choosing a truth assignment α for the variables. Finally, when it reads $\langle X \rangle$, A branches universally, picking one of the m clauses of φ in every branch. When it reads the suffix $\langle /X \rangle \langle /v_n \rangle \cdots \langle /v_1 \rangle$ of w , A tests that α makes the chosen clause true.

To this end, the automaton A uses three kinds of states:

- *assignment states*, q_+ and q_- , corresponding to true and false, respectively,
- *clause states* q_j , for $j \in [m]$, representing the clause chosen from φ to be tested for truth and
- a starting state q_0 and an accepting state q_F .

For the formal construction, let $\Phi = Q_1 x_1 \dots Q_n x_n \varphi$ be the input formula for QBF with $Q_i \in \{\exists, \forall\}$ for all $i \in [n]$ and a quantifier-free boolean formula $\varphi = C_1 \wedge \dots \wedge C_m$ with clauses C_j . Let w be constructed as above.

The ANWA $A = (Q, \Sigma, \delta, q_0, \{q_F\})$ in normal form is defined as follows:

- $Q = \{q_0, q_+, q_-, q_F\} \cup \{q_j \mid j \in [m]\}$;
- $\Sigma = \{v_i \mid i \in [n]\} \cup \{X\}$;
- For $q \in \{q_0, q_+, q_-\}$ and $i \in [n]$,

$$\delta(q, \langle v_i \rangle) = \begin{cases} q_+ \vee q_- & \text{if } Q_i = \exists, \\ q_+ \wedge q_- & \text{if } Q_i = \forall; \end{cases}$$
- $\delta(q, \langle X \rangle) = q_1 \wedge \dots \wedge q_m$;
- For $q \in \{q_+, q_-\}$ and $j \in [m]$,

$$\delta(q_j, q, \langle /X \rangle) = \begin{cases} q_F & \text{if } x_n \text{ occurs in } C_j \text{ and } q = q_+ \text{ or } \neg x_n \\ & \text{occurs in } C_j \text{ and } q = q_-, \\ q_j & \text{otherwise.} \end{cases};$$
- For $q \in \{q_+, q_-\}$, $j \in [m]$, and $2 \leq i \leq n$,

$$\delta(q_j, q, \langle /v_i \rangle) = \begin{cases} q_F & \text{if } x_{i-1} \text{ occurs in } C_j \text{ and } q = q_+ \text{ or} \\ & \neg x_{i-1} \text{ occurs in } C_j \text{ and } q = q_-, \\ q_j & \text{otherwise.} \end{cases}$$
- For all $q \in Q$, $\delta(q, q_0, \langle /v_1 \rangle) = q$, and
- For all $q \in Q$ and $i \in [n]$, $\delta(q_F, q, \langle /v_i \rangle) = q_F$.

It remains to be shown that Φ evaluates to true if and only if $w \in L(A)$.

We first note that A is deterministic on the suffix $\langle /X \rangle \langle /v_n \rangle \cdots \langle /v_1 \rangle$ of w . It is not hard to show that, on this suffix, A reaches the accepting state from state q_j , if and only if, the truth assignment α induced by the choices on $\langle v_1 \rangle \cdots \langle v_n \rangle$ makes C_j true. Thus, the subrun on the suffix $\langle X \rangle \langle /X \rangle \langle /v_n \rangle \cdots \langle /v_1 \rangle$ of w is accepting, if and only if, α makes all m clauses true. Finally, the existential and universal branching of A on $\langle v_1 \rangle \cdots \langle v_n \rangle$ corresponds to the quantification of the variables of Φ in the obvious and correct way. \square

Lemma 8 (restated). *Given a state $q \in Q$, an alphabet symbol $a \in \Gamma$, and $\mathcal{C}^k[G]$, for some $k \geq 1$, the call effect $\mathcal{C}^{k+1}[G](a, q)$ can be computed in doubly exponential time in $|G|$.*

Proof. Let $a \in \Sigma$, $q \in Q$, $X \subseteq Q$, and $k \geq 0$. We show that, given $\mathcal{C}^k[G]$ and a set $X \subseteq Q$, it can be decided in doubly exponential time in $|Q|$ and polynomial time in $|R|$ whether a subset of X is in $\mathcal{C}^{k+1}[G](a, q)$.

Let A_C be as defined for the proof of Lemma A.6 with $\mathcal{C}^k[G]$ as its basic Call effect, and let A be its modification with initial state q and set X of accepting states. A accepts all nested strings w on which there exists a strategy σ for JULIET of Call depth at most k such that $\text{states}_G(q, w, \sigma) \subseteq X$.

Let, for each $a \in \Gamma$, $A_a = (Q_a, \Sigma_a, \delta_a, q_{0,a}, F_a)$ be a NWA for R_a .

By definition, X has a subset in $\mathcal{C}^{k+1}[G](a, q)$, if JULIET has a strategy σ of call depth $k+1$ on $\langle a \rangle \langle /a \rangle$ that plays Call on $\langle /a \rangle$ and fulfils $\text{states}_G(q, \langle a \rangle \langle /a \rangle, \sigma) \subseteq X$. Such a strategy σ for JULIET exists if and only if for every word $w \in R_a$ there is a strategy σ_w of JULIET on w with $\text{states}_G(q, w, \sigma_w) \subseteq X$, thus if and only if $R_a \subseteq L(A)$, equivalently $R_a \cap \overline{L(A)} = \emptyset$.

By using a standard product construction and a complementation of an ANWA, the test boils down to a non-emptiness test for an ANWA with a state set of polynomial size in $|G|$ and can thus be done in doubly exponential time thanks to Proposition 7.¹⁸ \square

Adequacy of the fixed-point process

The following lemma will be used in the proof of Proposition 9.

Lemma A.7. For a cfG $G = (\Sigma, \Gamma, R, T)$ with a deterministic target NWA $A(T) = (Q, \Sigma, \delta, q_0, F)$, it holds, for every $a \in \Gamma$ and $q \in Q$:

$$\mathcal{C}[G](a, q) = \text{Mix}(\{\mathcal{E}[G, w](q) \mid w \in R_a\}).$$

Proof. Since both sides of the claimed equation are minimal sets, it suffices by Lemma A.4 to show that each element of a set on one side of the equation has a subset on the other side.

(\supseteq): Let $a \in \Gamma$, $q \in Q$ and $X \in \mathcal{C}[G](a, q)$. By definition of $\mathcal{C}[G]$, there exists a strategy $\sigma \in \text{STRAT}_{J, \text{Call}}$ such that $X = \text{states}_G(q, \langle a \rangle \langle /a \rangle, \sigma)$. Again by definition, JULIET plays Call on $\langle /a \rangle$ according to σ .

For every choice $w \in R_a$ with which ROMEO might respond to JULIET's initial Call move on $\langle /a \rangle$, there is a sub-strategy σ_w of σ on w . For each $w \in R_a$, let $X_w = \text{states}_G(q, w, \sigma_w)$. Obviously, each X_w has a subset in $\mathcal{E}[G, w](q)$, and therefore the set $X' = \bigcup_{w \in R_a} X_w$ has a subset in $\text{Mix}(\{\mathcal{E}[G, w](q) \mid w \in R_a\})$. It only remains to be proven that $X' \subseteq X$, so let $q' \in X'$. Then, by the definition of X' , there is some $w \in R_a$, strategy $\tau_w \in \text{STRAT}_R$ and $w' \in \text{WF}(\Sigma)$ such that $w' = \text{word}_G(w, \sigma_w, \tau_w)$ and $\delta^*(q, w') = q'$. From the way σ_w was defined from σ , it follows that $w' \in \text{words}_G(\langle a \rangle \langle /a \rangle, \sigma)$, and therefore $q' \in X$.

(\subseteq): Let $a \in \Gamma$, $q \in Q$ and $X \in \text{Mix}(\{\mathcal{E}[G, w](q) \mid w \in R_a\})$. Then, for each $w \in R_a$ there exists some set $X_w \in \mathcal{E}[G, w](q)$ such that $X = \bigcup_{w \in R_a} X_w$. By the definition of $\mathcal{E}[G, w]$, this means that for every $w \in R_a$ there is some strategy $\sigma_w \in$

¹⁸Note that the transition formulas of this ANWA may be of exponential size in $|G|$; however, the upper bound proof for the complexity of AVPA emptiness testing in [5] still yields only a doubly exponential time complexity in $|G|$ here.

STRAT such that $\text{states}_G(q, w, \sigma_w) = X_w$. Let $\sigma \in \text{STRAT}_{J, \text{Call}}$ be the strategy on $\langle a \rangle \langle /a \rangle$ where JULIET plays Call on $\langle /a \rangle$ and then, if ROMEO picks $w \in R_a$ as a replacement, keeps playing according to σ_w on w . By definition of $\mathcal{C}[G]$, the set $X' = \text{states}_G(q, \langle a \rangle \langle /a \rangle, \sigma)$ has a subset in $\mathcal{C}[G](q, a)$, and it only remains to be proven that $X' \subseteq X$. Let therefore $q' \in X'$. Then, there is some strategy $\tau \in \text{STRAT}_R$ and string $w' \in \text{WF}(\Sigma)$ such that $w' = \text{word}_G(\langle a \rangle \langle /a \rangle \sigma, \tau)$. Since JULIET's first move according to σ is a Call on $\langle /a \rangle$, there is some string $w \in R_a$ which ROMEO chooses as a replacement according to τ ; by definition of σ , it then holds that $q' \in \text{states}_G(q, w, \sigma_w) = X_w \subseteq X$ as was to be proven. \square

For the following proof, the *width* of a nested word is the maximum number of children of any node in its corresponding forest. Its *root width* is just the number of trees in its forest. The (*nesting*) *depth* of a nested word is the depth of its canonical forest representation.

Proposition 9 (restated). *For every cfG G it holds: $\mathcal{C}^*[G] = \mathcal{C}[G]$.*

Proof. For the proof we construct from a cfG $G = (\Sigma, \Gamma, R, T)$ a game $G' = (\Sigma, \Gamma, R', T)$, where R' consists of particular finite sublanguages $R'_a \subseteq R_a$, for every $a \in \Gamma$. Then we show

- (a) $\mathcal{C}^*[G] = \mathcal{C}^*[G']$,
- (b) $\mathcal{C}^*[G'] = \mathcal{C}[G']$, and finally
- (c) $\mathcal{C}[G'] = \mathcal{C}[G]$.

To construct G' , we first examine the algorithm from the proof of Lemma 8 more closely. For a given state $q \in Q$, alphabet symbol $a \in \Sigma$, state set $X \subseteq Q$ and effect $\mathcal{C}^k[G]$, the output of that algorithm depends only on the existence of a single string from R_a – for $a \in \Gamma$, the algorithm rejects if and only if there is a string in R_a that is *not* accepted by A . For each $q \in Q$, $a \in \Sigma$, $X \subseteq Q$ and $k \geq 1$, let $w(q, a, k, X)$ be one such *witness string* of minimum length, if such a string exists. Obviously, the output of the algorithm from Lemma 8 for input q, a, X and $\mathcal{C}^k[G]$ does not change if we replace R_a by any subset of R_a containing $w(q, a, k, X)$.

Let k^* be the smallest number with $\mathcal{C}^*[G] = \mathcal{C}^{k^*}[G]$, and let W_a be the set containing all $w(q, a, k, X)$ for all $q \in Q$, $k \leq k^*$ and $X \subseteq Q$. Furthermore, for each $w \in \text{WF}(\Sigma)$, let $v(a, w)$ be a string of minimum length such that $\mathcal{E}[G, w] = \mathcal{E}[G, v(a, w)]$ and $v(a, w) \in R_a$ and let $V_a = \{v(a, w) \mid w \in \text{WF}(\Sigma)\}$. Since there are only finitely many different string effects, each set V_a for $a \in \Sigma$ must be finite as well.

The replacement rules R' for G' are now constructed as follows: For each $a \in \Gamma$, let $R'_a \stackrel{\text{def}}{=} W_a \cup V_a$. By construction, it holds that R'_a is a finite subset of R_a , and an easy induction argument (along with the above considerations) shows that $\mathcal{C}^k[G'] = \mathcal{C}^k[G]$ for each $k \geq 1$. Along with the definition of $\mathcal{C}^*[\cdot]$, this proves (a).

For (b) it is sufficient to show that each finite strategy $\sigma \in \text{STRAT}_J[G']$ on a word w has bounded Call depth. This can be easily established with the help of König's Lemma. To this end, we consider the strategy tree $T_{\sigma, w}$ for σ on w where each node is a game position of the form (p, u, v) with a player index $p \in \{J, R\}$ and strings $u, v \in (\langle \Sigma \rangle \cup \langle / \Sigma \rangle)^*$ and each node corresponding to a game position κ has as children the possible follow-up positions κ' such that

$\kappa \xrightarrow{\sigma, \tau} \kappa'$ for σ and some counter-strategy $\tau \in \text{STRAT}_R$. Each node of this tree has a finite number of children – nodes corresponding to positions belonging to JULIET have only a single child each (as σ is fixed), and positions in which ROMEO is to replace some $a \in \Sigma$ have one child for each string in R'_a . Thus, the Call depth of nodes is bounded, as otherwise $T_{\sigma, w}$ would be a finitely branching tree with branches of arbitrary length, which by König's Lemma would yield that T has an infinite branch, contradicting the finiteness assumption for σ .

Towards (c), we prove the slightly stronger claim that $\mathcal{E}[G, w](q) = \mathcal{E}[G', w](q)$ for all $q \in Q$ and $w \in \text{WF}(\Sigma)$. Lemma A.7 then implies (c). To this end, we prove that each set in $\mathcal{E}[G', w](q)$ has a subset in $\mathcal{E}[G, w](q)$ and vice versa, which proves the desired equality by Lemma A.4.

One of these directions is almost trivial, as ROMEO simply has no more possible moves in G' than in G . Thus, any strategy $\sigma \in \text{STRAT}_{J, \text{Call}}(G)$ induces a sub-strategy $\sigma' \in \text{STRAT}_{J, \text{Call}}(G')$ with $\text{words}_{G'}(w, \sigma') \subseteq \text{words}_G(w, \sigma)$ and therefore also $\text{states}_{G'}(q, w, \sigma') \subseteq \text{states}_G(q, w, \sigma)$.

For the other direction, let $q \in Q$, $w \in \text{WF}(\Sigma)$ and let $\sigma' \in \text{STRAT}_{J, \text{Call}}(G')$ with $X = \text{states}_{G'}(q, w, \sigma') \in \mathcal{E}[G', w](q)$. Let $d \stackrel{\text{def}}{=} \text{Depth}^{G'}(\sigma', w)$. This is well-defined as σ' is finite. We prove by nested induction over $(d, \text{nesting depth of } w, \text{root width of } w)$ that there exists a strategy σ in G with $\text{states}_G(q, w, \sigma) \subseteq X$, which implies that X has a subset in $\mathcal{E}[G, w](q)$.

If $d = 0$, then JULIET only plays Read on the entirety of w ; obviously, this strategy is feasible in G as well and yields the same result.

If $d > 0$, JULIET must play Call on w at some point, and therefore it holds that $w \neq \epsilon$.

If $w = uv$ for $u, v \in \text{WF}(\Sigma)$, let σ'_u be the sub-strategy of σ' on u , and let $\{q_1, \dots, q_k\} = \text{states}_{G'}(q, u, \sigma'_u)$. For each $i \in [k]$, let further $\sigma'_{v,i}$ be a sub-strategy of σ' on v in case the play on u yields some string u' with $\delta^*(q, u') = q_i$. By induction (as u and v have smaller root width than w), there exist strategies σ_u on u and $\sigma_{v,i}$ on v in G such that $\text{states}_G(q, u, \sigma_u) \subseteq \{q_1, \dots, q_k\}$ and $\text{states}_G(q_i, v, \sigma_{v,i}) \subseteq \text{states}_{G'}(q_i, v, \sigma'_{v,i})$. Let σ be the strategy on uv in G where JULIET plays according to σ_u on u and according to $\sigma_{v,i}$ if the play on u yielded a string u' with $\delta^*(q, u') = q_i$. Then, it holds that $\text{states}_G(q, w, \sigma) \subseteq \bigcup_{i \in [k]} \text{states}_{G'}(q_i, v, \sigma'_{v,i}) \subseteq X$.

If $w = \langle a \rangle v \langle /a \rangle$ for some $a \in \Gamma$, $v \in \text{WF}(\Sigma)$, let $\delta(q, \langle a \rangle) = (q', p)$, let σ'_v be the sub-strategy of σ' on v , and let $\{q_1, \dots, q_k\} = \text{states}_{G'}(q', v, \sigma'_v)$. By induction (as the depth of v is smaller than the depth of w), there exists a strategy σ_v on v in G with $\text{states}_G(q', v, \sigma_v) \subseteq \text{states}_{G'}(q', v, \sigma'_v)$. In the strategy σ on w , JULIET plays according to σ_v on v . The play on v from q' according to σ is bound to reach some state q_i for $i \in [k]$. If there is some string $v_i \in \text{words}_{G'}(v, \sigma'_v)$ with $\delta^*(q', v_i) = q_i$ such that JULIET would play Read on $\langle /a \rangle$ according to σ' in G' if the play on v yields v_i , then JULIET also plays Read on $\langle /a \rangle$ according to σ ; obviously, in this case, the resulting state from the play according to σ is in X . Otherwise, JULIET plays Call on $\langle /a \rangle$ in σ . Let $z \in R_a$ be some arbitrary response for ROMEO to this Call move in G ; we now explain how JULIET plays on z according to σ .

By construction, the replacement language R'_a in G' contains the string $v(a, z)$, so this string is a valid response for ROMEO to the Call by JULIET on $\langle /a \rangle$ in G' . Let $\sigma'_{v(a,z)}$ be the sub-strategy of σ' if ROMEO chooses this response. As $\sigma'_{v(a,z)}$ has a Call depth of at most $d - 1$, by induction there exists

a strategy $\sigma_{v(a,z)}$ for JULIET on $v(a,z)$ in G with $\text{states}_G(q, v(a,z), \sigma_{v(a,z)}) \subseteq \text{states}_{G'}(q, v(a,z), \sigma'_{v(a,z)})$. By the definition of $v(a,z)$, it holds that $\mathcal{E}[G, z] = \mathcal{E}[G, v(a,z)]$, which implies that there is a strategy σ_z for JULIET on z in G such that $\text{states}_G(q, z, \sigma_z) \subseteq \text{states}_G(q, v(a,z), \sigma_{v(a,z)})$. In σ , JULIET then plays on z according to σ_z , and the above set inclusions show that all states resulting from this play are in X as well, which completes the case $w = \langle a \rangle v \langle /a \rangle$ for $a \in \Gamma$ and concludes the proof. \square

Lower bounds

Similar to Lemma A.6, where we constructed ANWA from given cfGs to obtain upper complexity bounds, we prove matching lower bounds for by transforming ANWA into cfGs.

Lemma A.8. There is a polynomial time algorithm that computes, given an ANWA A and a nested word w , a cfG $G = (\Sigma, \emptyset, \Gamma, R, T)$ and a nested word w' such that $w \in L(A)$ if and only if ROMEO has a winning strategy on w' in G , against all replay-free strategies of JULIET. Furthermore, G only depends on A (not on w) and can be computed in polynomial time in the size of A .

Proof. Let $A = (Q, \Sigma, q_0, \delta, \{q_F\})$ be an ANWA and $w \in \text{WF}(\Sigma)$ a nested word. The idea is to simulate the alternation of A in the game G on w' . We design G to only admit replay-free strategies for JULIET. To make this possible, we construct w' from w by adding substrings that offer enough “space” for this simulation.

We assume without loss of generality that Σ does not contain any symbols from $(Q \times Q) \cup Q \cup \{b, 0, 1, \vee, \wedge, \perp, \top\}$. For any formula $\varphi \in \mathcal{B}^+(Q \times Q) \cup \mathcal{B}^+(Q)$, the *encoding* $\text{Enc}(\varphi)$ is the well-nested string over the alphabet $Q \cup \{\vee, \wedge\}$ derived from φ in the natural way:

- If $\varphi \in \{\perp, \top\}$, then $\text{Enc}(\varphi) = \langle \varphi \rangle \langle / \varphi \rangle$;
- If $\varphi = (q, p) \in Q \times Q$, then $\text{Enc}(\varphi) = \langle (q, p) \rangle \langle / (q, p) \rangle$;
- If $\varphi = q \in Q$, then $\text{Enc}(\varphi) = \langle q \rangle \langle / q \rangle$;
- If $\varphi = \varphi_1 \vee \varphi_2$, then $\text{Enc}(\varphi) = \langle \vee \rangle \langle / \vee \rangle \langle b \rangle \text{Enc}(\varphi_1) \text{Enc}(\varphi_2) \langle / b \rangle$;
- If $\varphi = \varphi_1 \wedge \varphi_2$, then $\text{Enc}(\varphi) = \langle \wedge \rangle \langle / \wedge \rangle \langle b \rangle \text{Enc}(\varphi_1) \text{Enc}(\varphi_2) \langle / b \rangle$.

Let q_1, \dots, q_m be an enumeration of the states in Q .

Let Σ' and Σ'' be two distinct copies of Σ with symbols of the form a' and a'' , respectively, for every $a \in \Sigma$.

For each $a \in \Sigma$, we define

- $v(\langle a \rangle) \stackrel{\text{def}}{=} \langle a' \rangle \langle / a' \rangle \text{Enc}(\delta(q_1, \langle a \rangle)) \cdots \text{Enc}(\delta(q_m, \langle a \rangle)) \langle a \rangle$, and
- $v(\langle / a \rangle) = \langle a'' \rangle \langle / a'' \rangle \text{Enc}(\delta(q_1, q_1, \langle / a \rangle)) \cdots \text{Enc}(\delta(q_m, q_m, \langle / a \rangle)) \langle / a \rangle$.

We note that in $v(\langle a \rangle)$, for each $i \leq m$, there is a subword $\text{Enc}(\delta(q_1, \langle a \rangle))$, whereas in $v(\langle / a \rangle)$ there is a subword $\text{Enc}(\delta(q_i, q_j, \langle / a \rangle))$, for every $i, j \leq m$. The string w' is defined as the nested word $v(w)$, that results from w by replacing every tag $\sigma \in \langle \Sigma \rangle \cup \langle / \Sigma \rangle$ with $v(\sigma)$.

As explained above, the purpose of the game G is to simulate the alternation of A . We associate existential branching with ROMEO and universal branching with JULIET.¹⁹ To this end, the replacement languages for $\langle \vee \rangle$ and $\langle \wedge \rangle$ are as follows.

- $R_\vee = \{\langle 1 \rangle \langle /1 \rangle, \langle 2 \rangle \langle /2 \rangle\};$
- $R_\wedge = \{\langle 2 \rangle \langle /2 \rangle\};$

All other symbols should not be replaced in the game, so we set $\Gamma = \{\vee, \wedge\}$.

The intention of the construction is that the behaviour of A on w is simulated as follows in the game on $v(w)$ in G . Choices corresponding to \vee -gates in transitions are taken by ROMEO (and we force JULIET to call every symbol \vee as strings containing \vee -tags will not be accepted by the target NWA). The choice of $\langle 1 \rangle \langle /1 \rangle$ by ROMEO is interpreted by the choice of the first branch of the formula by A and likewise for $\langle 2 \rangle \langle /2 \rangle$ and the second branch. Choices corresponding to \wedge -gates in transitions are taken by JULIET: we interpret $\langle \wedge \rangle \langle / \wedge \rangle$ just as $\langle 1 \rangle \langle /1 \rangle$ in the \vee -case. Therefore, if JULIET reads $\langle \wedge \rangle \langle / \wedge \rangle$ this corresponds to choosing the first branch of the formula, if she calls it, she chooses the second branch.

The target automaton follows the choices taken by the two players. At opening tags of the form $\langle (q, p) \rangle$ it interprets q and p as the next horizontal and hierarchical state, respectively. It accepts if it ends in an accepting state or reaches $\langle \top \rangle \langle / \top \rangle$ at some point. If it reaches $\langle \perp \rangle \langle / \perp \rangle$ at some point, it rejects.

It remains to show that indeed w is accepted by A if and only if ROMEO has a winning strategy on $v(w)$ in G .

We call a strategy for JULIET on $v(w)$ *valid* if JULIET plays Call on every \vee symbol. Since JULIET can never win with a strategy that is not valid, we restrict our attention to valid strategies for JULIET on $v(w)$.

We will now show that each run of A on w corresponds to some strategy τ of ROMEO on $v(w)$ in G , and that an accepting run on w induces a winning strategy on $v(w)$ and vice versa.

Let τ be a strategy for ROMEO on $v(w)$, and let σ be some tag in w . We say that a subformula φ' encoded in $v(\sigma)$ is *enabled* according to τ and some counterstrategy for JULIET if the resulting sub-play on $v(\sigma)$ yields a substring of the form $\langle 1 \rangle \langle /1 \rangle \langle b \rangle \text{Enc}(\varphi') \text{Enc}(\psi) \langle /b \rangle$ or $\langle 2 \rangle \langle /2 \rangle \langle b \rangle \text{Enc}(\psi) \text{Enc}(\varphi') \langle /b \rangle$ (for some formula ψ). By the construction of $v(\sigma)$, for each $q \in Q$ (and $\gamma \in \Gamma$, if $\sigma \in \langle / \Sigma \rangle$) the set of all states $q' \in Q$ such that q' might be enabled in the sub-play on $\text{Enc}(\delta(q, \sigma))$ (resp. $\text{Enc}(\delta(q, \gamma, \sigma))$) according to τ and some valid counterstrategy for JULIET satisfies the formula $\delta(q, \sigma)$ (resp. $\delta(q, \gamma, \sigma)$). In this way, the strategy τ induces a run ρ of A on w such that for each valid counter-strategy of JULIET, the resulting rewriting of $v(w)$ corresponds to one path in ρ .

Similarly, a run ρ of A on w induces a strategy τ for ROMEO on $v(w)$; if, for some tag σ in w and state $q \in Q$, $P \subseteq Q \times Q$ (resp. $P \subseteq Q$) is the follow-up state set satisfying $\delta(q, \sigma)$ (resp. $\delta(q, p, \sigma)$ for some appropriate $p \in Q$), τ can be constructed to enable exactly the states from P for all counter-strategies of JULIET.

As the target automaton in G accepts a rewriting of $v(w)$ if and only if it encodes a path in a run of A on w ending in an accepting state, the correspon-

¹⁹The reader might feel that it would be more natural to associate existential moves to JULIET. Why our chosen association is useful will become clear in the proof of Proposition 10 given below.

dence between runs of A on w and strategies of ROMEO on $v(w)$ in G implies that there exists a winning strategy for ROMEO on $v(w)$ in G if and only if there is an accepting run of A on w . \square

Using Lemma A.8, it is easy to prove our lower bounds.

Proposition 10 (restated). *For the class of unrestricted games JWIN is*

(a) 2-EXPTIME-hard with bounded replay, and

(b) PSPACE-hard with no replay.

Proof. The proof that $\text{JWIN}^k(\mathcal{G}_{\text{all}})$ is 2-EXPTIME-hard for all $k \geq 2$ is by a reduction from the emptiness problem for ANWA, which is 2-EXPTIME-hard according to Proposition 7(a).

Given an ANWA A , let G' be the VP-cfG constructed by the algorithm of Lemma A.8. Let G be the game with an additional new function symbol s which ROMEO is allowed to rewrite by any string of the form $v(w)$ as defined in the proof of Lemma A.8. Then $L(A)$ is non-empty if and only if ROMEO has a winning strategy on $\langle s \rangle / s$ in G . This yields the desired reduction from emptiness for ANWA to $\text{JWIN}^2(\mathcal{G}_{\text{all}})$.

PSPACE-hardness of $\text{JWIN}^1(\mathcal{G}_{\text{all}})$ follows directly from the corresponding hardness result for the ANWA membership problem (Prop. 7 (b)) along with the existence of a polynomial-time reduction proven in Lemma A.8. \square

Finite replacement languages

Proposition 11 (restated). *For the class of unrestricted games with finite replacement languages, $\text{JWIN}(\mathcal{G})$ is*

(a) EXPTIME-complete with unbounded replay, and

(b) PSPACE-complete with bounded or without replay.

Proof. As already mentioned in the body of the paper, the lower bounds follow from Theorem 4.3 in [14] and the proof of Proposition 10. Thus, only the upper bounds need to be established.

For (a), the non-emptiness test for $R_a \cap \overline{L(A)}$ can be replaced by a membership test $v \in \overline{L(A)}$, for each of the finitely many strings $v \in R_a$. This can be done in polynomial space by Proposition 7. The exponential time upper bound then immediately follows because the number of iterations of the fixpoint process is at most exponential and the final test whether w is accepted by $A_{\mathcal{C}[G]}$ needs only exponential time.

For (b), a polynomial space algorithm for a bounded number k of replay works basically just as in the general case, by first computing the call effect $\mathcal{C}^k[G]$ from the input game G , then computing from it the ANWA $A_{\mathcal{C}}^k$ from Proposition 6 and finally simulating $A_{\mathcal{C}}^k$ on the input string w . The initial call effect, $\mathcal{C}^1[G]$, can again be computed in polynomial time. For each i , $\mathcal{C}^{i+1}[G]$ can be computed from $\mathcal{C}^i[G]$ in polynomial space and finally, whether $A_{\mathcal{C}}^k$ accepts w can be tested in polynomial space in $|w|$ and the number of states of $A_{\mathcal{C}}^k$, that is, in the number of states of the target automaton of G .

Some care is needed though, as the (representation of the) intermediate automata and the resulting automaton $A_{\mathcal{C}}^k$ can be of superpolynomial (at most

exponential) size. However, as usual for space bounded computations, the information about A_C^k and the intermediate automata can be recomputed whenever it is needed. The composition of these constantly many polynomial space computations then yields an overall polynomial space bound. It is crucial here that, as observed in the proof of Proposition 7, the evaluation of A_C^k is possible in polynomial space in $|w|$ and the number of states of A_C^k . \square

By a more complicated argument, the upper bound of Proposition 11 (a) can even be established in the case where the finite replacement languages are not given explicitly but by NWAs.

Proofs for Section 4

For our upper bounds, we formalise XML Schema²⁰ target languages by way of simple NWA (as defined in Section 4) and use similar techniques as in the upper bound proofs for Section 3. Lower bounds, on the other hand, will generally follow from lower bounds for context-free games on *flat* strings, as defined in [14].

Upper bounds

The general structure of the algorithms is the same as in Section 3. Technically, the two main parts of the proof are to show that SANWAs are suitable (SNWAs can be computed from XML Schemas, Proposition 15, and SANWAs from (simple) game effects, Proposition A.9) and to establish the complexity of SANWAs (Propositions A.11 and 17).

Suitability of simple NWAs

First off, we prove that simple NWA are at least as expressive as single-type tree grammars. The idea behind this is rather straightforward, as we only need to combine DFAs for each type's content model and, on reading some opening tag $\langle a \rangle$, start some DFA in a sub-computation to check the nested string between $\langle a \rangle$ and the associated $\langle /a \rangle$ for compliance with the content model of some type X . Thanks to the single-type property, the type X is uniquely defined by a and the context from which $\langle a \rangle$ was read, so we obtain a deterministic automaton as desired.

Proposition 15 (restated). *From every single-type tree grammar T , a simple DNWA A can be computed in polynomial time, such that $L(A) = L(T)$.*

Proof. Let $T = (\Sigma, \Delta, S, P, \lambda)$ be a single-type tree grammar. We will construct a SNWA A such that $L(T) = L(A)$.

Due to the single-type property, for each type $X \in \Delta$ and each $a \in \Sigma$, there is at most one type X' in the content model of X with $\lambda(X') = a$. Without loss of generality, assume that there is exactly one such type for each X and a (which can be done by adding a “dummy type” X_\perp with $r_{X_\perp} = \emptyset$ to T), and denote this type by $\nu(X, a)$.

For each $X \in \Delta$, let $A_X = (P_X, \Delta, \delta_X, p_{0,X}, F_X)$ be a DFA deciding $L(r_X)$ (which can be computed from the *deterministic* regular expression r_X in polynomial time). Assume w.l.o.g. that all P_X, P_Y are disjoint for $X \neq Y$. Then, the SNWA $A = (Q, \Sigma, \delta, (p_0, 0), \{(p_f, 0)\})$ is defined as follows:

- $Q = \{\perp\} \cup P \times \Delta'$, with
 - $P = \{p_0, p_f\} \cup \bigcup_{X \in \Delta} P_X$ and
 - $\Delta' = \Delta \cup \{0\}$, with $0 \notin \Delta$
- δ is defined by
 - $\delta((p_0, 0), \langle \lambda(S) \rangle) = (q_{0,S}, S)$,
 - $\delta((p, X), \langle a \rangle) = (p_{0,\nu(X,a)}, \nu(X, a))$ for each $a \in \Sigma, p \in P, X \in \Delta$,

²⁰For more background on formalisations of XML Schema we refer the reader to [10].

- $\delta(q, q', \langle a \rangle)$ is defined by t below as per the definition of SNWA,
- $F_{\text{loc}}(a) = \bigcup_{X \in \Delta: \lambda(X)=a} (F_X \times \{X\})$, and
- t is defined by
 - $t((p_0, 0), \lambda(S)) = (q_f, 0)$ and
 - $t((p, X), a) = (\delta_X(p, a), X)$ for each $a \in \Sigma$, $p \in P$, $X \in \Delta$.

To show that $L(T) = L(A)$, it suffices to show that for every $w \in \text{WF}(\Sigma)$ and $X \in \Delta$, it holds that $\langle \lambda(X) \rangle w \langle \lambda(X) \rangle \in L(X)$ if and only if $\delta^*((q_0, X), w) \in F_{\text{loc}}(\lambda(X))$, where $L(X)$ is defined like $L(T)$ with root type X . The claimed equality then follows with $L(T) = L(S)$. \square

Proposition A.9. There is an algorithm that computes from the call effect $\mathcal{C}[G]$ of a simple game G in polynomial time in $|\mathcal{C}[G]|$ and $|G|$ a SANWA A_C such that $L(A_C) = \text{JWin}(G)$.

Proof. We construct A_C^ℓ almost as the automaton A_C in the proof of Proposition 6. However, as they are mimicking games, the alternating transitions in A_C occur at closing tags, whereas the definition of simple ANWAs requires that alternating transitions occur only at opening tags. Thus we slightly adapt the construction as follows.

Let $A(T) = (Q, \Sigma, \delta, q_0, F_0)$ be a SNWA in normal form and let $P, \Delta, F_{\text{loc}}, t, \perp$ witness the simplicity of $A(T)$. With $q? \notin Q$, we let $A_C^\ell \stackrel{\text{def}}{=} ((Q \cup \{q?\}), \Sigma, \delta_C^\ell, q_0, F_0)$, where δ_C^ℓ is defined as follows

- For every $q \in Q$ and $a \in \Sigma$, where $q' = \delta(q, \langle a \rangle)$,

$$\delta_C^\ell(q, \langle a \rangle) \stackrel{\text{def}}{=} ((q', t(q, a)) \wedge (q', q?)) \vee \bigvee_{X \in C(q)} \bigwedge_{r \in X} (q', r).$$

- For every $q, q' \in Q$ and $a \in \Sigma$, $\delta_C^\ell(q, q', \langle a \rangle)$ is defined via a target state function t_C as per the definition of SANWA.

The target state function t_C and final state function $F_{\text{loc}, C}$ witnessing the simplicity of A_C^ℓ are defined by $t_C(q, a) \stackrel{\text{def}}{=} q$ and $F_{\text{loc}, C}(a) \stackrel{\text{def}}{=} F_{\text{loc}}(a)$, respectively. This automaton obviously fulfils both simplicity conditions, by construction and the simplicity of $A(T)$. The correctness of the automaton is proven analogously to the proof of Lemma A.6. \square

Complexity of simple ANWAs

To prove the upper bound in Proposition 17 (a), i.e., that non-emptiness for SANWAs is in PSPACE, we start off by proving a somewhat stronger result: That the problem of determining, given a NWA A and a SANWA B , whether there is a nested word accepted by both A and B , is in PSPACE. The standard approach for proving a result of this sort (a product construction between two NWA or two SANWA) is generally not feasible here, as SANWA are less expressive than NWA (so A cannot in general be transformed into a SANWA) and transforming B into a NWA might incur a doubly exponential blow-up in size. Therefore, a PSPACE algorithm has to be constructed especially for this problem and uses the following pumping property for strings in $L(A) \cap L(B)$.

As in the previous section, the *width* of a nested word is the maximum number of children of any node in its corresponding forest. Its *root width* is just the number of trees in its forest. The *depth* of a nested word is the depth of its canonical forest representation.

Lemma A.10. Let $A = (Q_A, \Sigma, \delta_A, q_{0,A}, F_A)$ be a NWA and let $B = (Q_B, \Sigma, \delta_B, q_{0,B}, F_B)$ be a SANWA with type alphabet Δ , final state function F_{loc} , target state function t and test state $q_? \in Q$. Then $L(A) \cap L(B) \neq \emptyset$ if and only if there exists a string in $L(A) \cap L(B)$ of width at most $2^{|Q_B|} \cdot |\Sigma| \cdot |Q_A|$ and depth at most $3(|\Sigma| + 1)|Q_A|^2|\Delta|$.

Proof. The “if” direction is trivial. For “only if”, assume for the sake of contradiction that $L(A) \cap L(B) \neq \emptyset$, but there is no string in $L(A) \cap L(B)$ fulfilling the claimed upper bounds on both width and depth.

First, we observe that for all words $w \in L(B)$, all nodes of any depth i in an arbitrary accepting run of B on w contain only linear states of the same type, i.e. if $\rho = (D, \lambda)$ is an accepting run of B on w , and $x, y \in D$ with $|x| = |y| = i$ for any $i \in \mathbb{N}$, and if $\bar{\lambda}(x) = (p, X)$ and $\bar{\lambda}(y) = (p', Y)$, then $X = Y$. This can be proven by a simple induction on i .

In the remainder of this proof, if ρ is a run of B on some string w and ρ' is a sub-run of ρ on a nested substring w' of w , we call ρ' *successful* if all leaves of ρ' are accepting with respect to the context of w' , i.e. if all leaves of ρ' are in F in case $w' = w$, or if all leaves of ρ' are in $F_{\text{loc}}(a)$ in case $\langle a \rangle w' \langle /a \rangle$ is a substring of w . Note that due to the definition of runs, all test subruns of ρ' (i.e. subruns starting with horizontal state $q_?$) have to accept. Furthermore, by the above observation, all subtrees of ρ' immediately below its root start from the same state, as that state is uniquely given by the tags enclosing w' and the root type of ρ' .

First off, let $w \in L(A) \cap L(B)$ be a string of width greater than $2^{|Q_B|} \cdot |\Sigma| \cdot |Q_A|$ and minimal length. We now prove that $L(A) \cap L(B)$ contains a string shorter than w , in contradiction to the assumed minimality.

Let w' be a maximum-length nested substring of w with root width greater than $2^{|Q_B|} \cdot |\Sigma| \cdot |Q_A|$. Let ρ be an accepting run of B on w and ρ' its sub-run on w' . Similarly, since A may also be viewed as an ANWA, there is an accepting run π of A on w in which each non-leaf node has only a single child. Let π' be the sub-run of π on w' . For $k = 1, \dots, |w'|$, let $k\text{-layer}(w') \in \Sigma \times ((\mathcal{P}(Q_B) \times Q_A) \cup (\mathcal{P}(Q_B^2) \times Q_A^2))$ such that if $w'_k = \langle a \rangle$, $(q_1, p_1), \dots, (q_\ell, p_\ell)$ are all pairs of states at depth k in ρ' and (q, p) is the state pair of depth k in π' , then $k\text{-layer}(w') = (a, \{(q_1, p_1), \dots, (q_\ell, p_\ell)\}, (q, p))$, and if $w'_k = \langle /a \rangle$, $(q_1), \dots, (q_\ell)$ are all states at depth k in ρ' and q is the state at depth k in π' , then $k\text{-layer}(w') = (a, \{q_1, \dots, q_\ell\}, q)$. As the root width of w' is greater than $|\Sigma \times \mathcal{P}(Q_B) \times Q_A|$, there are numbers $i < j < |w'|$ such that $i\text{-layer}(w') = j\text{-layer}(w')$ and the substrings $w'_1 \dots w'_i$ and $w'_1 \dots w'_j$ (and therefore also $w'_{i+1} \dots w'_j$) are well-nested. The claim, then, is that there are accepting runs of A and B on the string \tilde{w} derived from w by deleting $w'_{i+1} \dots w'_j$ from w' .

Assume now, again for the sake of contradiction, that there is no string in $L(A) \cap L(B)$ of width at most $2^{|Q_B|} \cdot |\Sigma| \cdot |Q_A|$ and depth at most $3(|\Sigma| + 1)|Q_A|^2|\Delta|$. By the above part of the proof, this means that all strings fulfilling the requirement on width must be of a depth exceeding $3(|\Sigma| + 1)|Q_A|^2|\Delta|$. Let w be such a string of minimal length, and let ρ be an accepting run of B and π an accepting run of A on w .

As the nesting depth of w is greater than $3(|\Sigma| + 1)|Q_A|^2|\Delta|$, there exist well-nested strings w' and w'' such that for some $a \in \Sigma$,

- $\langle a \rangle w' \langle /a \rangle$ is a substring of w ,
- $\langle a \rangle w'' \langle /a \rangle$ is a substring of w' ,
- all sub-runs of ρ on w' and w'' start from the same state $q_a \in Q_B$
- either all sub-runs of ρ on w' and w'' are unsuccessful or there exist successful runs in ρ on both w' and w'' , and
- the states of A according to π before and after reading $\langle a \rangle w' \langle /a \rangle$ are the same as those before and after reading $\langle a \rangle w'' \langle /a \rangle$.

The claim is that both A and B have accepting runs on the string \tilde{w} derived from w by replacing w' with w'' . As w'' is a proper substring of w' , proving this claim yields the desired contradiction to the minimal length of w and thus the claim of Lemma A.10 \square

Proposition A.11. There is an alternating algorithm that tests in polynomial time whether, for an NWA A and a SANWA B it holds $L(A) \cap L(B) \neq \emptyset$.

Proof. We formulate the claimed algorithm as a game for two players, whom we will call ADAM and EVE to avoid confusion with the players for context-free games. This game will always terminate after at most polynomially many rounds, so an alternating polynomial-time algorithm can easily be constructed from it by branching nondeterministically (resp. universally) for the moves for EVE (resp. ADAM) and accepting the input if and only if EVE wins.

We will construct the game such that that EVE has a winning strategy on input NWA $A = (Q_A, \Sigma, \delta_A, q_{0,A}, F_A)$ and SANWA $B = (Q_B, \Sigma, \delta_B, q_{0,B}, F_B)$ with final state function F_{loc} , test state $q_?$ and target state function t if and only if $L(A) \cap L(B) \neq \emptyset$. EVE's goal in this game is to prove that there is a string that is accepted by both A and B without writing down that string explicitly; by Lemma A.10, it does suffice to examine strings of at most exponential width and polynomial depth, but such a string can still not be explicitly spelled out using only polynomial space. We therefore represent a string implicitly by the behaviour it induces in A and B .

Game positions for EVE consist of two states $p_1, p_2 \in Q_A$, a function $S : Q_B \rightarrow \mathcal{P}(Q_B)$ and two numbers $c, n \geq 0$, and the game is constructed in such a way that EVE has a winning strategy from position (q_1, q_2, S, c, n) if and only if there is a string w of root width at most 2^c and nesting depth at most n such that $q_1 \xrightarrow{w}_A q_2$, and for every $q \in Q_B$ there is a run of B on w beginning in q and ending inside $S(q)$. We write c_0 for $|Q_B| \log(|\Sigma| \cdot |Q_A|)$ and n_0 for $3(|\Sigma| + 1)|Q_A|^2|\Delta|$. Lemma A.10 then guarantees that $L(A) \cap L(B)$ is nonempty if and only if there is a state $q_f \in F_A$ and a function S with $S(q_{0,B}) \subseteq F_B$ such that EVE has a winning strategy from position $(q_{0,A}, q_f, S, c_0, n_0)$.

In any position (q_1, q_2, S, c, n) , EVE has the following options:

- If $c > 0$, she may choose to play a *concatenation round*, asserting that $w = v_1 v_2$ for strings $v_1, v_2 \in \text{WF}(\Sigma)$ whose root width is at most half that of w . In this case, she chooses two functions $S_1, S_2 : Q_B \rightarrow \mathcal{P}(Q_B)$, corresponding to strings v_1, v_2 as above and an “in-between” state $q' \in$

Q_A . The functions S_1 and S_2 have to fulfil the condition that for each $q \in Q_B$ it holds that $S(q) = \bigcup_{p \in S_1(q)} S_2(p)$; if S_1 and S_2 do not fulfil this condition, ADAM wins. Otherwise, ADAM has a choice of which part of EVE's assertion he wants to contest, so he may choose as a follow-up position either $(q_1, q', S_1, c-1, n)$ or $(q', q_2, S_2, c-1, n)$.

- If $n > 0$, EVE may choose to play a *nesting round* (with $a \in \Sigma$), asserting that $w = \langle a \rangle v \langle /a \rangle$ for some $v \in \text{WF}(\Sigma)$. To this end, she first chooses an alphabet symbol a and a function S' corresponding to v as above, as well as states $q'_1, q'_2, p \in Q_A$ such that $\delta_A(q_1, \langle a \rangle) = (q'_1, p)$ and $\delta_A(q'_2, p, \langle /a \rangle) = q_2$ (if no such states exist, ADAM wins immediately). Next, ADAM chooses some $q \in Q_B$ on which to contest EVE's claim. In response, EVE picks a state $p' \in Q_B$ and a set of states $P = \{p_1, \dots, p_k\} \subseteq Q_B$ such that $(\{p'\} \times P) \models \delta_B(q, \langle a \rangle)$ and $S(q) = \bigcup_{p \in P \setminus \{q?\}} \{t(p, a)\}$. If she cannot choose such a set, ADAM wins.

If ADAM has not won by this point, he has to contest EVE's claim that there is a string v such that B has a successful run on v . If $q? \in P$ and $S'(p') \not\subseteq F(a)$, the string v claimed by EVE fails the test subrun mandated by B branching with $q?$, so in this case, ADAM wins. Otherwise, the game continues from position $(q'_1, q'_2, S', c_0, n-1)$, as the root width of the substring v is bounded by 2^{c_0} .

- EVE may choose to *solve* (with $a \in \Sigma$), asserting that $w = \langle a \rangle \langle /a \rangle$. In this case, she chooses a symbol $a \in \Sigma$. Similar to a nesting round, ADAM then picks a state $q \in Q_B$ on which to contest EVE's claim, to which EVE responds by choosing a state $p \in Q_B$ and a set $P \subseteq Q_B$. The game then ends and a winner is determined. EVE wins if and only if all of the following conditions are fulfilled:
 - (a) There are states $p', q' \in Q_A$ such that $\delta_A(q_1, \langle a \rangle) = (q', p')$ and $\delta_A(q', p', \langle /a \rangle) = q_2$;
 - (b) $(\{p\} \times P) \models \delta_B(q, \langle a \rangle)$;
 - (c) $S(q) = \bigcup_{p \in P \setminus \{q?\}} \{t(p, a)\}$;
 - (d) If $q? \in P$, then $p \in F(a)$.
- EVE may choose to *solve with* ϵ , asserting that $w = \epsilon$. In this case, the game ends and EVE wins if and only if $q_1 = q_2$ and for each $q \in Q_B$ it holds that $S(q) = \{q\}$.

Since each round that does not end the game decreases either the number of remaining nesting or concatenation rounds and the number of remaining concatenation rounds only increases at the end of a nesting round, the total number of rounds starting from $(q_{0,A}, q_f, S, c_0, n_0)$ is bounded by $c_0 n_0$, which is polynomial in the size of A and B . It is easy to see that each choice by EVE or ADAM requires only a polynomial-size certificate, and that each check for winning conditions is computable in polynomial time. Therefore, an alternating algorithm checking whether EVE has a winning strategy on this game (as described above) has a polynomial upper bound on its running time. It remains to be shown that this algorithm indeed tests A and B for intersection emptiness, i.e. that EVE has a winning strategy from $(q_{0,A}, q_f, S, c_0, n_0)$ for some $q_f \in F_A$ if and only if $L(A) \cap L(B) \neq \emptyset$.

To prove this claim, we show that the following statements are equivalent:

- (1) EVE has a winning strategy from position (q_1, q_2, S, c, n) ;
- (2) There is a string $w \in \text{WF}(\Sigma)$ of width at most $2^{|Q_B|}|\Sigma||Q_A|$, root width at most 2^c and depth at most n such that there is a run of A on w from q_1 to q_2 , and for each $q \in Q$, there is a successful run of B on w from q ending inside $S(q)$.

(1) \Rightarrow (2): Assume EVE has a winning strategy σ from position (q_1, q_2, S, c, n) . We prove (2) by induction on the structure of σ .

If EVE solves with ϵ as her first move according to σ , the string $w = \epsilon$ obviously fulfils the claim of (2).

If EVE's first move according to σ is to solve with some $a \in \Sigma$, then $w = \langle a \rangle \langle /a \rangle$ fulfils the claim of (2). Since $c, n \geq 0$, w fulfils the desired upper bounds on nesting depth and width; winning condition (a) ensures the existence of a run of A ; and as for each $q \in Q$ that ADAM chooses, EVE can respond with a set of horizontal states compliant with the transition formulae of B according to winning conditions (b) to (d), the desired runs of B on w exist as well.

If EVE begins with a concatenation round according to σ , it follows that there exist a state $q' \in Q_A$ and functions $S_1, S_2 : Q_B \rightarrow \mathcal{P}(Q_B)$ such that for each $q \in Q_B$ it holds that $S(q) = \bigcup_{p \in S_1(q)} S_2(p)$ and EVE has a winning strategy on both $(q_1, q', S_1, c-1, n)$ and $(q', q_2, S_2, c-1, n)$. By induction, this implies that there are strings $v_1, v_2 \in \text{WF}(\Sigma)$ of width at most $2^{|Q_B|}|\Sigma||Q_A|$, root width at most 2^{c-1} and depth at most n for which there exist appropriate runs of A and B ; it is easy to see that $w = v_1 v_2$ fulfils the width and depth requirements of the claim, and that the claimed runs of A and B on w can be constructed by combining those on v_1 and v_2 .

If EVE starts by playing a nesting round with some $a \in \Sigma$, there exists a function S' as well as states $q'_1, q'_2, p \in Q_A$ such that $\delta_A(q_1, \langle a \rangle) = (q'_1, p)$ and $\delta_A(q'_2, p, \langle /a \rangle) = q_2$. Furthermore, for each $q \in Q_B$, there is a state $p \in Q_B$ and a set $P \subseteq Q_B$ such that $(\{p\} \times P) \models \delta_B(q, \langle a \rangle)$ and $S(q) = \bigcup_{p \in P \setminus \{q\}} \{t(p, a)\}$, and if $q \in P$ then $S'(p) \subseteq F(a)$. Finally, EVE has a winning strategy starting from position $(q'_1, q'_2, S', c_0, n-1)$.

By induction, there exists a string v of width at most $2^{|Q_B|}|\Sigma||Q_A|$ and depth at most $n-1$ for which there exist appropriate runs of A and B ; the string $w = \langle a \rangle v \langle /a \rangle$ therefore fulfils the claimed restrictions on depth and width. Again, it is easy to see that a run of A on w can be constructed from the one on v . To construct the desired runs of B on w , denote the run on v starting at p and ending in $S'(p)$ by ρ and let $q \in Q_B$. A run on w starting at q is then constructed as follows: The root node, labelled q , has $\{p\} \times P$ as the set of labels of its children. Each of these nodes (p, p') is the root of a copy of ρ , whose leaves are all inside $S'(p)$; if $p' = q$, the leaves of the corresponding copy of ρ have no further children; otherwise, their only child is labelled with the state $t(p', a)$. Using the above properties and the definition for SANWA semantics, it is easy to verify that the tree thus constructed is indeed a run of B on w starting at q and ending inside $S(q)$.

(2) \Rightarrow (1): This part of the proof is by an induction on the structure of w analogous to the above proof of (1) \Rightarrow (2). \square

Proposition 17 (restated).

(a) *Non-emptiness for SANWA is PSPACE-complete.*

(b) *The membership problem for SANWA is decidable in polynomial time.*

Proof. That non-emptiness for SANWAs is in PSPACE follows directly from Proposition A.11, as alternating polynomial time equals polynomial space.

PSPACE-hardness can be proven by a simple reduction (with a constant-sized NWA A accepting $\text{WF}(\Sigma)$) from the nonemptiness problem for SANWA, which in turn is PSPACE-hard by reduction from the nonemptiness problem for alternating finite automata, interpreting flat strings $w_1 \dots w_n \in \Sigma^*$ as nested strings $\langle w_1 \rangle \langle /w_1 \rangle \dots \langle w_n \rangle \langle /w_n \rangle \in \text{WF}(\Sigma)$ of nesting depth 0 (and vice versa). It is then quite easy to construct from an AFA B' a SANWA B such that $L(B') \neq \emptyset$ if and only if B accepts some nested string of depth 0.

Together, statement (a) follows.

To show (b), that the membership problem for SANWAs can be decided in polynomial time, it suffices to show that the problem can be decided by an alternating Turing machine with logarithmic space. The computation of a SANWA A can be easily simulated by an alternating Turing machine M . To this end, the TM M could branch existentially and universally, just as A . In particular, on a word w it would have exactly one run for each run of A on w . However, such a naive simulation would need to remember the stack contents to compute $t(p, a)$ at the next closing tag $\langle /a \rangle$, and thus the space required would be proportional to the nesting depth of the input word.

To achieve a logarithmic space bound, we can modify M as follows. Whenever a transition at an opening tag $\langle a \rangle$ yields a pair (q, p) with $p \neq q?$, the computation branches universally into two subcomputations: one moves directly to the corresponding closing tag $\langle /a \rangle$ and continues after that from state $t(p, a)$. The other proceeds as A on the current subword but does *not* need to remember p . Whenever such a computation reaches a closing tag it accepts. Test subruns, starting from a pair $(q, q?)$ are simulated slightly different: they remember the nesting depth of the opening tag $\langle a \rangle$ and behave at the corresponding closing tag just as A would. However, if a test subrun starts a test-subsubrun the latter only needs to remember the new nesting depth, as it can stop when the subsubrun has finished.

The correspondence between runs of A and the ATM can be shown by induction on the nesting depth of the input word w . In particular, the ATM accepts just if A does.

More formally, we claim that Algorithm 1 evaluates a SANWA $A = (Q, \Sigma, \delta, q_0, F)$ with local acceptance function F_{loc} , test state $q?$ and target state function t on a nested word $w = w_1 \dots w_n \in \text{WF}(\Sigma)$ (with $w_i \in \langle \Sigma \rangle \cup \langle / \Sigma \rangle$ for each $i \in [n]$). To this end, it keeps track of a current state $q \in Q$ of A , two indices i and j denoting the starting and ending position in w of the substring to be verified in its current run, and an index $f \in \Sigma \uplus \{0\}$ that tracks whether the current string is to be verified against the accepting states of A ($f = 0$) or some $F_{\text{loc}}(a)$ ($f = a$). To simplify notation for the former case, we let $F_{\text{loc}}(0) \stackrel{\text{def}}{=} F$.

We first elaborate on how to execute line 7 of Algorithm 1 in alternating logarithmic space. Assume that each transition function $\delta(q, \langle a \rangle)$ in A is given in prefix notation, i.e. formulas are of the form (i) $\wedge(\varphi_1, \varphi_2)$ or (ii) $\vee(\varphi_1, \varphi_2)$ or (iii) (q', p) . In case (i), the algorithm guesses universally whether to branch

Algorithm 1 $\text{Verify}(A, w)$

```
1:  $q \leftarrow q_0$ 
2:  $i \leftarrow 1$ 
3:  $j \leftarrow n$ 
4:  $f \leftarrow 0$ 
5: while  $i \leq j$  do
6:   //  $i$  always denotes the position of an opening tag
7:   Choose alternatingly  $(q', p)$  according to  $\delta(q, w_i)$ 
8:   if  $p \neq q'$  then
9:      $i \leftarrow (\text{position of closing tag associated with } w_i) + 1$ 
10:     $q \leftarrow t(p, w_i)$ 
11:   else
12:     //  $p = q'$ ; start test subrun:
13:      $q \leftarrow q'$ 
14:     if  $w_{i+1} \in \langle \Sigma \rangle$  then
15:        $i \leftarrow i + 1$ 
16:        $j \leftarrow (\text{position of closing tag associated with } w_i)$ 
17:     else
18:       //  $w_{i+1}$  is the associated closing tag of  $w_i$ ; end test subrun and exit
       loop to test for acceptance.
19:        $j \leftarrow i$ 
20:        $i \leftarrow i + 1$ 
21:   if  $q \in F_{\text{loc}}(f)$  then
22:     Accept
23:   else
24:     Reject
```

into φ_1 or φ_2 , in case (ii) this choice is existential, and in case (iii), a result is fixed. Clearly, this is feasible in alternating logarithmic space and equivalent to first choosing existentially a set $P \subseteq Q^2$ with $P \models \delta(q, \langle a \rangle)$ and then universally picking a tuple $(q', p) \in P$.

It is also easy to see that Algorithm 1 terminates (as the value of i increases in each iteration of the loop in line 5 while j only ever decreases) and requires only logarithmic space.

It remains to be proven that Algorithm 1 is correct. We do this by proving that, for any nested word w , state $q \in Q$, $f \in \Sigma \cup \{0\}$ and indices i, j such that $w_i \dots w_j$ is a well-nested string, lines 5-24 of Algorithm 1 accept in an alternating fashion if and only if there is a run of A on $w_i \dots w_j$ starting at q and ending inside $F_{\text{loc}}(f)$. The proof is by induction on the structure of w and uses as its crucial component the above insight that picking a follow-up state tuple from $\delta(q, w_i)$ in line 7 is equivalent to universally selecting a child of a depth i node labelled q in some run of A , and that each existential strategy for the alternating execution of Algorithm 1 corresponds to a single run of A in this way. \square

Lower bounds

All of our lower bounds for simple games follow from lower bounds for cfGs on flat strings, that is, games on strings as defined in [14], with target and replacement languages represented by deterministic regular expressions. Lower bound results for replay-free games and bounded replay with finite replacement languages and target languages represented as DFAs were already proven in [14]. They can be transferred to games with target languages described by deterministic regular expressions. As an entirely new result compared to [14], we prove here a lower bound for bounded replay (actually, Call depth 2 suffices) and later sketch how these results carry over to nested word cfGs.²¹

Intuitively, a regular expression is deterministic, if each of its positions can be matched uniquely with a symbol of the regular expression, without lookahead. Formally let, for a regular expression r , $D(r)$ be the expression, in which the i -th symbol σ of r is replaced by (σ, i) , e.g. $D((a+b)^*a) = ((a, 1) + (b, 2))^*(a, 3)$. We call r is *deterministic*, if there do not exist strings w, v, v' , symbol σ and numbers i, j such that $w(\sigma, i)v \in L(D(r))$, $w(\sigma, j)v' \in L(D(r))$ and $i \neq j$.

Lemma A.12. For the class of games on flat strings with target and replacement languages specified by deterministic regular expressions, JWIN is PSPACE-hard with bounded replay of Call depth 2.

Proof. We prove this by reduction from the complement of the problem CORRIDOR TILING:

Given a set U of *tiles*, relations $V, H \subseteq U \times U$ of *vertical* and *horizontal constraints*, *initial* and *final* tiles $u_i, u_f \in U$ and a number n (represented in unary), is there a correct tiling of width n and arbitrary height that starts with u_i , ends with u_f and violates none of the (vertical or horizontal) constraints.

Formally, a *tiling* of width n and height m is a mapping $t : [n] \times [m] \rightarrow U$. A tiling t is *valid* if

- $t(0, 0) = u_i$,
- $t(n, m) = u_f$,
- for every $i \in [n - 1]$ and $j \in [m]$, $(t(i, j), t(i + 1, j)) \in H$, and
- for every $i \in [n]$ and $j \in [m - 1]$, $(t(i, j), t(i, j + 1)) \in V$.

CORRIDOR TILING asks whether an instance $\mathcal{I} = (U, u_i, u_f, V, H, n)$ has a valid tiling of width n . It is well known that this problem is PSPACE-complete (see, e.g., [7] for a slightly different definition of tilings). Since PSPACE is closed under complementation, the complement of CORRIDOR TILING is complete for PSPACE as well.

We give here a reduction from the complement of CORRIDOR TILING to JWIN.

The reduction constructs, given an instance $\mathcal{I} = (U, u_i, u_f, V, H, n)$ for CORRIDOR TILING, a game $G = (\Sigma, \Gamma, R, T)$ and a symbol s from Γ such that JULIET has a winning strategy on s if and only if \mathcal{I} does *not* have a valid corridor tiling. The basic idea is that, after JULIET's first Call move on s , ROMEO will answer with an

²¹Note that our PSPACE lower bound for bounded replay is not in conflict with the corresponding PTIME upper bound in [14]. This is because the PTIME upper bound given there required replacement languages to be finite, whereas we consider here replacement languages given by arbitrary deterministic regular expressions, which may be infinite.

encoding w of a valid corridor tiling, if one exists. With her Call moves of depth 2, JULIET may then try to flag inconsistencies (i.e. constraint violations) in the tiling given by ROMEO; finally, the target automaton should accept a tiling if JULIET did indeed point out an actual inconsistency.

The game G is over an alphabet Σ which is obtained by the union of U with a set \hat{U} of disjoint copies \hat{u} of all elements $u \in U$ and the set $\{s, ?_h, ?_v, !_h, !_v, \#\}$ for some $s, ?_h, ?_v, !_h, !_v, \# \notin U$. We set $\Gamma = U \cup \{s, ?_h, ?_v\}$. The replacement and target languages are described below.

A *tiling candidate* (for \mathcal{I}) is a string of the form $((U?_v?_h)^n\#)^*$, whose length- n blocks of elements from U are supposed to be interpreted as lines of a tiling, with *protest symbols* $?_v, ?_h$ after each tile and a *line separator* symbol $\#$ at the end of each line. The replacement language R_s consists of all tiling candidates v such that u_i is the first symbol of v . It is easy to see that R_s can be described by a DRE of polynomial size in $|\mathcal{I}|$. The other replacement languages are very simple: $R_u = \{\hat{u}\}$ for each $u \in U$, $R_{?_h} = \{!_h\}$ and $R_{?_v} = \{!_v\}$.

The construction of the target language T is best motivated by sketching how plays can proceed on the input string s . First, JULIET should be forced to play Call on s and allow ROMEO to actually give a candidate for a valid tiling. Therefore, $s \notin T$.

By the definition of R_s , ROMEO responds to this Call with a tiling candidate which already begins with the correct tile. It is now JULIET's task to flag an error in this tiling, i.e. either

- two tiles separated by $(n - 1)$ tiles with corresponding protest symbols and one line separator symbol (a potential vertical error), or
- two tiles separated by exactly 2 protest symbols (a potential horizontal error), or
- a single tile at the end of v (a potential incorrect final tile).

To flag any tiles, JULIET plays Call on them, forcing ROMEO to replace any called tile x by a *marked tile* \hat{x} . If the marked tiles indeed make up an error, we want JULIET to win, so the DRE for T should describe such strings. If, on the other hand, JULIET tries to cheat by marking too few or too many tiles, or tiles that do not make up an error, she should lose the game.

To allow easy DRE-based checking of the three types of errors mentioned above, JULIET also has to specify the *type* of error right after the first tile she flagged; in case of a horizontal (vertical) error, she has to Call $?_h$ ($?_v$) to have it replaced with $!_h$ ($!_v$). This basically “tells” the target DRE what sort of error to check for. An incorrect final tile does not need its own type of protest symbol, because (as we will see) a flagged inconsistency of this sort can be recognised by a DRE “as is”.

To construct the target language DRE, we first define some abbreviations:

- For any set $S = \{s_1, \dots, s_k\}$ and REs α_s for each $s \in S$, $\bigoplus_{s \in S} \alpha_s$ stands for the RE $\alpha_{s_1} + \dots + \alpha_{s_k}$;
- U' denotes the DRE $\bigoplus_{u \in U} u?_v?_h$, and $(U' + \#)^k$ the k -fold repetition of $(U' + \#)$;
- for each $u \in U$, $V_u \stackrel{\text{def}}{=} !_v?_h(U' + \#)^n(\bigoplus_{(u, u') \notin V} \hat{u}')?_v?_h(U' + \#)^*$;

- for each $u \in U$, $H_u \stackrel{\text{def}}{=} !_h(\bigoplus_{(u,u') \notin V} \hat{u}')?_v?_h(U' + \#)^*$;

It is easy to verify that for each $u \in U$, V_u , and H_u are DREs of polynomial size in $|\mathcal{I}|$. Intuitively, V_u (H_u) describes all suffixes immediately to the right of \hat{u} ($\hat{u}?_v$) in tilings where JULIET has correctly flagged a vertical (horizontal) error starting with u .

The target language T can now be described by the DRE

$$(U' + \#)^* (\hat{u}_f(V_{u_f} + ?_v H_{u_f}) + \bigoplus_{u \in U \setminus \{u_f\}} \hat{u}(V_u + ?_v(H_u + ?_h \#))),$$

which is also of polynomial size in $|\mathcal{I}|$. It is easy to see that if a valid tiling exists, ROMEO can simply win the game by providing it in the first move. Therefore, in this case, JULIET does *not* have a winning strategy. On the other hand, if no tiling exists, ROMEO can only give a tiling candidate with at least one (vertical, horizontal or final tile) error in his first move and JULIET can win by marking one such error. \square

The following two results can be shown by careful adaptation of the corresponding lower bound proofs in [14].

Lemma A.13. For the class of games on flat strings with target and replacement languages specified by deterministic regular expressions, JWIN is PTIME-hard (under logspace reductions) without replay.

Lemma A.14. For the class of games on flat strings with target and replacement languages specified by deterministic regular expressions, JWIN is EXPTIME-hard with unlimited replay.

Proposition 18 (restated). *For the class of games with target languages specified by DTDs, JWIN is*

- (a) EXPTIME-hard with unrestricted replay,
- (b) PSPACE-hard with bounded replay, and
- (c) PTIME-hard (under logspace-reductions) without replay

Proof. All lower bounds follow by the same reduction from corresponding lower bounds for flat cfGs, which were just given as Lemma A.14, Lemma A.12 and Lemma A.13.

The idea for the reduction from flat cfGs to simple (nested) cfGs is as follows: All input and replacement strings $w = w_1 \dots w_n \in \Sigma^*$ are replaced by $\hat{w} = \langle w_1 \rangle \langle /w_1 \rangle \dots \langle w_n \rangle \langle /w_n \rangle \in \text{WF}(\Sigma)$; to this end a target DFA $A(T)$ is simulated by a SNWA in normal form with an extra state q_n such that $\delta(q, \langle a \rangle) = q_n$ for each q and a , $F_{\text{loc}}(a) = q_n$ for each a , and $t(q, a)$ is the transition function of $A(T)$. Replacement NFAs are similarly transformed into NWAs. \square

Using the reduction from the proof of Proposition 18, Theorem 12 also yields the following result, which we will need in later proofs:

Corollary A.15. For the class of games on flat strings with target languages specified by DFAs, JWIN is PTIME-complete without replay.

Proposition 19 (restated). *For the class of games with target languages specified by XML Schemas and explicitly enumerated finite replacement languages, JWIN is*

- (a) EXPTIME-complete with unrestricted replay, and
- (b) PTIME-complete (under logspace-reductions) with bounded replay or without replay.

The same results hold for DTDs in place of XML Schemas.

Proof. The upper bound in (a) follows from Theorem 12. The lower bounds in (a) and (b) follow from Lemma A.14 and Lemma A.13, respectively, as there the replacement rules are finite. It thus only remains to show the upper bound in (b).

The proof is quite similar to the proof of the upper bound in Proposition 17 (b). It combines an alternating logspace-computation, that simulates all plays of the game on the input string, with universal branching to divide, at each opening tag $\langle a \rangle$, the processing of the remaining word into the processing of the subword until the corresponding closing tag $\langle /a \rangle$ and the processing of the remaining word after that $\langle /a \rangle$.

We first describe how the game on an input string w can be simulated by an alternating logspace-computation. This part of the proof is very similar to the proof of the upper bound of Theorem 5.8 in [14]. Let k be the bound on the replay depth. We consider the equivalent version of cfGs in which JULIET decides already when she reads an opening tag $\langle a \rangle$, whether she wants ROMEO to rewrite a subword $u = \langle a \rangle \cdots \langle /a \rangle$.

The idea is that the choices of JULIET and ROMEO are simulated by existential and universal branching of the algorithm in the obvious fashion. However, if JULIET calls an opening tag $\langle a \rangle$ at some position i and ROMEO replaces the corresponding subword $u = \langle a \rangle \cdots \langle /a \rangle$ of w by a word v from R_a then the algorithm does not actually replace u but rather stores the information that u has been replaced by a pointer to i and another pointer to v (which is stored in the representation of G). As the replay depth is bounded by k , at each time at most k such pairs of pointers are active, consuming at most $\mathcal{O}(\log(|G|))$ many bits. The test whether the resulting word (of each branch) is accepted by the target automaton T is integrated into this branching process as follows. Each process maintains a current linear state p reflecting the state of T in the unique computation on the prefix of the current string, that is, if the current game configuration is (J, w_1, w_2) , the current state is the one obtained by T after reading w_1 . Whenever JULIET reads an opening tag $\langle a \rangle$, the computation universally branches into two subcomputations. The first subcomputation checks whether JULIET has a winning strategy in the game on the subword between $\langle a \rangle$ and its corresponding closing tag. The other subcomputation continues after that closing tag in the state determined by the target state function. JULIET can only win if both subcomputations accept. Each subcomputation may recursively branch in the same way. When a subcomputation reaches a closing tag $\langle /a \rangle$ it accepts if the current linear state is in $F_{\text{loc}}(a)$ and rejects otherwise. It is not hard to see that this algorithm has an accepting run on a word w if and only if JULIET has a winning strategy on w . As the algorithm only uses logarithmic space it witnesses the desired PTIME upper bound. \square

Proofs for Section 5

In this section, we give proofs for our results concerning parameter validation and games with insertion stated in Section 5.

Validation of parameters

In this subsection, we consider cfGs with *parameter validation*, i.e. games of the form $G = (\Sigma, \Gamma, R, V, T)$ which have an additional *validity relation* $V \subseteq \Gamma \times \text{WF}(\Sigma)$. We will generally assume each *validation language* $V_a \stackrel{\text{def}}{=} \{w \in \text{WF}(\Sigma) \mid (a, w) \in V\}$ (for $a \in \Gamma$) to be a nonempty nested word language conforming to some specification (e.g. NWA, DTD or XML Schema). The semantics of such games is similar to the general semantics for cfGs, except for the fact that, in a configuration $(J, u\langle a \rangle v, \langle /a \rangle w)$, JULIET is only able to play Call on $\langle /a \rangle$ if it holds that $\langle a \rangle v \langle /a \rangle \in V_a$. Note that, while it isn't strictly necessary to pass the outermost a to V_a along with v , we still do so in order to easier describe V_a as a language of trees with root node labelled a .

As mentioned in Section 5, we restrict our attention to games without replay, as we are seeking to identify tractable cases, and JWIN is already PSPACE-hard for bounded-replay games with target languages specified by DTDs *without* parameter validation.

Upper bounds

First off, we prove tractability for a restricted class of validation cfGs. As notation used in the proof, we say that a function symbol g is “from V_f ”, if $V_g = V_f$.

Theorem 20 (restated). *For the class of games with validation with a bounded number of validation DTDs and target languages specified by DTDs, JWIN is in PTIME without replay.*

Proof. (sketch) The basic proof idea for this result follows a similar approach to that used in [12]: Going through the input string (interpreted as a tree) in a bottom-up fashion, we check for each node's child string whether it (and the subtree below it) can be rewritten to fit the target and verification languages in a replay-free manner. This allows us to tell whether JULIET is able to safely play Read or Call on the node whose child string we just examined, and possibly on ancestor nodes as well. In this manner, deciding JWin(G) basically boils down to performing a polynomial number of safe rewritability tests for replay-free games on flat strings, which are each feasible in polynomial time by Corollary A.15.

For the sake of simple presentation, we identify trees and their nested word linearisations throughout this proof.

As described above, our goal is to subsequently remove subtrees in a bottom-up manner and only consider flat strings of leaf node labels. More precisely, each removal step replaces a subtree of depth one, that is, a node v whose children are all leaves, by a single node with a label that contains all relevant information about its (former) subtree with respect to the game. If, for instance, the subtree below a node v with function symbol f cannot be rewritten to conform to the corresponding part of some DTD V_f , this information will be encoded into the label of v and JULIET will never be able to play Call on v or any of its ancestors

with a function symbol from V_f , no matter her rewriting capabilities on other parts of the input tree.

Let t be the tree representing some well-nested rooted²² word w . By $\text{label}(v)$ we denote the label of a node v . By S we denote the set $\{T, V_1, \dots, V_d\}$ of schemas of the game. The *profile* $P(t') \subseteq S$ of a tree t' is the set of schemas for which t' is valid. We first consider subgames on subtrees t_v rooted at some node v with label a . With each replay-free strategy σ on t_v that does *not* play Call on v itself, we associate the *profile set* $\mathcal{P}_\sigma(t_v)$ of profiles P , for which ROMEO has a counterstrategy yielding a tree t' with $P = P(t')$. The *dossier* $\mathcal{D}(v)$ of v is the set of all sets X , for which there is a strategy σ of JULIET such that $\mathcal{P}_\sigma(t_v) \subseteq X$. In our words, $\mathcal{D}(v)$ is the closure of the set of all sets $\mathcal{P}_\sigma(t_v)$ under taking supersets.²³

In the bottom-up computation mentioned above, we plan to replace the subtree below each node v with label a and change v 's label to $(a, \mathcal{P}_\sigma(t_v))$. Once, this process reaches the root $\text{root}(t)$ of the tree, it can be instantly decided whether JULIET has a winning strategy on w . Indeed, this is the case if and only if $\mathcal{D}(\text{root}(t))$ contains a profile set \mathcal{P} , such that every profile $P \in \mathcal{P}$ contains the target schema T .

To illustrate the above definitions, we consider the special case $d = 1$, that is, besides the target schema T there is only one validation schema V . In this case, there are four possible profiles of trees: $\{V, T\}$, $\{V\}$, $\{T\}$, \emptyset . As an example, a tree has profile $\{V\}$ if it is valid with respect to V but not with respect to T .

The four different profiles yield $2^4 = 16$ possible profile sets and $2^{16} = 65536$ candidate dossiers. However, only the following six cases need to be distinguished:

- $\{\{V, T\}\} \in \mathcal{D}$: JULIET has a strategy that guarantees to yield a tree t' that is valid with respect to both schemas;
- $\{\{T\}\} \in \mathcal{D}$ and $\{\{V\}\} \in \mathcal{D}$: JULIET has a strategy that guarantees a tree t' in T and a strategy that guarantees a tree in t'' in V , but neither t' nor t'' is valid with respect to the other schema;
- $\{\{T\}\} \in \mathcal{D}$, but $\{\{V\}\} \notin \mathcal{D}$: JULIET has a strategy that guarantees a tree t' in T , but no strategy that guarantees a tree in t'' in V ;
- $\{\{V\}\} \in \mathcal{D}$, but $\{\{T\}\} \notin \mathcal{D}$: JULIET has a strategy that guarantees a tree t'' in V , but no strategy that guarantees a tree in t' in T ;
- $\{\{V\}, \{T\}\} \in \mathcal{D}$: JULIET has a strategy that guarantees to yield a tree that is either in T or in V , but she can not enforce either of the two;
- $\mathcal{D} = \{\{\emptyset\}\}$: no matter how JULIET plays, ROMEO can always enforce a tree that is invalid for both T and V .

In all lower cases, we assume that none of the upper cases applies.

We now start with the detailed description of the algorithm. We assume²⁴ that all content models of DTDs are given by DFAs.

²²For simplicity, we do not consider non-rooted words in this proof. They can be handled similarly.

²³The reason why we do not aim just at the set of all sets $\mathcal{P}_\sigma(t_v)$ will become clearer below.

²⁴As content models are given by *deterministic* regular expressions, these DFAs can be computed efficiently.

As stated above, the algorithm works in a bottom-up fashion. First, for all leaf nodes, their dossier is computed. As there is no actual subgame on a leaf node v (that does not play Call on that node), each such dossier is just $\{\{P(t_v)\}\}$. In this case, $P(t_v)$ is just the set of schemas in which the (original) label of v is allowed at a leaf node.

The key step that the algorithm performs is to compute the dossier of a node v with children u_1, \dots, u_m all of whose dossiers are already given. The idea is to compute $\mathcal{D}(v)$ with the help of replay-free games on flat strings, whose winning problem can be decided thanks to Corollary A.15.

For these flat games, the algorithm needs to compute, in a preprocessing phase that only depends on G , flat replacement sets R'_f , for every function symbol $f \in \Gamma$. As replacement strings represent strings in which no further Call moves are possible, the labels of their positions do not include dossiers but rather the profile of the actual tree that they represent.

Each set R'_f can be computed as follows. Let L_f denote the content model of f in V_f (represented by some DFA A_f). For each symbol a occurring in L_f , let $\Sigma_{f,a}$ be the set of all pairs (a, P) , such that there is a tree t' with profile P and root label a that is valid with respect to R_f . For each f , a and P , it can be decided in polynomial time whether $(a, P) \in \Sigma_{f,a}$ by constructing a deterministic tree automaton that accepts all trees that are valid with respect to R_f and the schemas in P , and invalid with respect to the schemas in $S \setminus P$. As d is fixed, this amounts to an emptiness test for the polynomial-size product of $d+1$ deterministic tree automata. It follows that all sets $\Sigma_{f,a}$ can be computed in time polynomial in the size of G .²⁵

The set R'_f consists of all strings over $\bigcup_{a \in \Sigma} \Sigma_{f,a}$ whose Σ -projection is in L_f . Given the sets $\Sigma_{f,a}$, a DFA for R'_f can be easily (and efficiently) computed.

Now, with the schemas R'_f at hand, we describe the computation of $\mathcal{D}(v)$ from u_1, \dots, u_m and their dossiers in more detail.

For a dossier $\mathcal{D} = \{\mathcal{P}_1, \dots, \mathcal{P}_\ell\}$ and a symbol a , let $s(a, \mathcal{D})$ denote the string²⁶ $(a, \mathcal{D})(a, \mathcal{P}_1) \cdots (a, \mathcal{P}_\ell) \cdot \#_a$.

The idea behind the construction of the flat game is as follows.

The original game on a tree t_z with root label g (where z is a child of the current root node v) can be viewed as follows: JULIET chooses a strategy for the first phase of the game before the closing tag $\langle/g\rangle$ of z is reached. This strategy corresponds to some profile set $\mathcal{P}_i \in \mathcal{D}(z)$. By choosing a counterstrategy for this subgame, ROMEO basically picks a profile $P \in \mathcal{P}_i$. Then JULIET decides whether she plays Call at $\langle/g\rangle$ (subject to validity with respect to V_g) and ROMEO replaces z , in case she plays Call.

In the flat game on $(g, \mathcal{D})(g, \mathcal{P}_1) \cdots (g, \mathcal{P}_\ell) \#_g$ this is mimicked as follows: JULIET chooses her strategy by playing Call at (g, \mathcal{P}_i) . ROMEO replaces (g, \mathcal{P}_i) by some pair (g, P) with $P \in \mathcal{P}_i$. So far the games exactly mimicks the original game *before* reaching $\langle/g\rangle$. If P allows JULIET to play Call at $\langle/g\rangle$ (that is, if $V_g \in P$), she can call the follow-up symbol $\#_g$ which is then replaced by ROMEO with a string from R'_g . The case that JULIET cheats by playing Call although $V_g \notin P$ can be easily detected by the target automaton (whose construction will be explained soon, otherwise).

²⁵Since the number of validation schemas, and thus also $|S|$, is fixed, the fact that we need superpolynomial time in $|S|$ is of no consequence here.

²⁶The order of the profile sets in $s(a, \mathcal{D})$ is inessential. We can assume just some ordering of profile sets.

For each of the $2^{2^{d+1}}$ possible profile sets \mathcal{Q} , the algorithm determines the winner for a particular replay-free game on the string $s(\text{label}(u_1), \mathcal{D}(u_1)), \dots, s(\text{label}(u_m), \mathcal{D}(u_m)))$ with replacement sets

- R'_a , for every symbol $\#_a$ and
- $\{(a, P_1), \dots, (a, P_j)\}$, for each symbol (a, \mathcal{P}) with $\mathcal{P} = \{(a, P_1), \dots, (a, P_j)\}$.

It only remains to specify the target language of the game, which, of course, depends on \mathcal{Q} . The DFA $A_{\mathcal{Q}}$ for the target language for profile set \mathcal{Q} has to determine whether JULIET has a winning strategy in the (original) subgame on t_v that yields a tree with a profile in \mathcal{Q} .

To this end, $A_{\mathcal{Q}}$ ignores all symbols that do not represent actual subtrees in the original game, that is,

- all symbols (g, \mathcal{D}) , as they only indicate the beginning of a substring for some node;
- all symbols (a, \mathcal{P}) with profile sets \mathcal{P} as they correspond to strategy options for JULIET that she did not choose; and
- all symbols $\#_g$ as they represent cases in which JULIET played Read and the respective subtree is represented by the symbol (g, P) , chosen by ROMEO;

We call all other symbols *relevant*.

Thus, $A_{\mathcal{Q}}$ accepts all strings y resulting from the game, for which the subsequence y' of relevant symbols is consistent with some profile $P \in \mathcal{Q}$. That is, if²⁷

- for all symbols (q, P') of y' it holds $P \subseteq P'$ and,
- for each schema $D \in P$ the Σ -projection of y' is in (the language of) D .

As d is fixed, $A_{\mathcal{Q}}$ is of polynomial size.

This completes the construction of the flat game and thus of the algorithm.

Each of the bottom-up reduction steps amounts to a (large but) constant number of tests whether JULIET has a winning strategy in a flat game without replay and therefore can be done in overall polynomial time.

It is not too difficult but tedious to verify that the algorithm is also correct. \square

Lower bounds

In this subsection, we prove lower bounds for less restricted classes of validation cfGs. We prove the lower bounds of Theorem 21 as single results in the order in which they were stated in Section 5: from most expressive to least expressive target, replacement and validation languages.

Theorem A.16. For the class of validation games with target, validation and replacement languages specified by DNWAs, JWIN is EXPTIME-hard without replay. This lower bound already holds for games with one single function symbol.

²⁷As we did not require that $\mathcal{D}(v)$ consists exactly of all profile sets $\mathcal{P}_{\sigma}(t_v)$, we do not need to ensure anything for profiles not in \mathcal{Q} .

Proof. We show EXPTIME-hardness by reduction from the intersection emptiness problem for deterministic nested word automata: Given n DNWAs A_1, \dots, A_n , does it hold that $L(A_1) \cap \dots \cap L(A_n) = \emptyset$? That this problem is EXPTIME-hard follows directly from the EXPTIME-hardness of the intersection emptiness problem for deterministic top-down finite tree automata [16].

Given DNWAs A_1, \dots, A_n over an alphabet Σ , we construct a game G and input string w such that JULIET has a winning strategy on w in G if and only if there is no string $v \in \text{WF}(\Sigma)$ accepted by all n automata. The game G uses the alphabet $\Sigma \cup \{s, t\}$, with $s, t \notin \Sigma$, and t being the only function symbol of G .

The input string is $w = \langle t \rangle^{n+1} \langle s \rangle \langle /s \rangle \langle /t \rangle^{n+1}$, i.e. the tree linearised by w is simply a path of length $n+2$ whose $n+1$ non-leaf nodes are labelled t and whose leaf is labelled s . According to G , play on w should proceed as follows: First, JULIET plays Call on the first $\langle /t \rangle$ in w (i.e. the innermost t). We emphasize that t is the only function symbol and is therefore used for two different purposes in this proof.

ROMEO replies to this call by providing some string $v \in \text{WF}(\Sigma)$; if possible, ROMEO will want to choose as v a string contained in the intersection of all $L(A_i)$ for $i \in [n]$. JULIET, in turn, will try to show that there is some $i \in [n]$ such that $v \notin L(A_i)$; she does so by playing Call on the i -th remaining $\langle /t \rangle$ in the rewritten string $\langle t \rangle^n v \langle /t \rangle^n$. The validation language for t will ensure that this Call is only possible if v is indeed not in $L(A_i)$. ROMEO can reply to such a Call by JULIET with an arbitrary string in $\text{WF}(\Sigma)$. However, the actual choice of this string is inconsequential as all is needed for JULIET to win is that there are less than n occurrences of $\langle /t \rangle$ in the resulting string.

More formally, the game G over alphabet $\Sigma \cup \{s, t\}$ with $\Gamma = \{t\}$ is defined with the replacement language $R_t = \text{WF}(\Sigma)$ and validation language $V_t = \{\langle t \rangle \langle s \rangle \langle /s \rangle \langle /t \rangle\} \cup \{\langle t \rangle^i v \langle /t \rangle^i \mid v \in \text{WF}(\Sigma) \setminus L(A_i), i \in [n]\}$. The target language is $T = \{\langle t \rangle^i v \langle /t \rangle^i \mid v \in \text{WF}(\Sigma), 0 \leq i < n\}$.

G can be efficiently computed from A_1, \dots, A_n , as DNWAs can be complemented in polynomial time, and given DNWAs for $\text{WF}(\Sigma) \setminus L(A_i)$ for each $i \in [n]$, DNWAs for V_t and T can easily be constructed.

Clearly, any strategy σ for JULIET on w in G that does not Call the first $\langle /t \rangle$ cannot be a winning strategy, as the target language does not contain any s tags and the only part of the validation language containing s tags only ever applies to the innermost t . From there, it is straightforward to prove that JULIET has a winning strategy on w if and only if ROMEO can not respond to this first Call with a string that is contained in the intersection of all $L(A_i)$ for $i \in [n]$, i.e. iff $L(A_1) \cap \dots \cap L(A_n) = \emptyset$. \square

Theorem A.17. For the class of validation games with target, validation and replacement languages specified by XML Schemas, JWIN is PSPACE-hard. This lower bound already holds for games with one single function symbol, whose replacement and target language are given by DTDs and whose replacement language is finite.

Proof. (sketch) We prove this claim by giving a reduction from the problem QBF of determining for a given quantified Boolean formula Φ , whether that Φ is true. We assume that the input formula is of the form $\Phi = Q_1 x_1 \dots Q_n x_n \varphi(x_1, \dots, x_n)$ with $Q_i \in \{\exists, \forall\}$ for all $i \in [n]$ and a Boolean formula $\varphi = C_1 \vee \dots \vee C_m$ with

m clauses in *disjunctive* normal form. Without loss of generality, we further assume that no clause contains both x_i and $\neg x_i$ for any $i \in [n]$.

We construct from Φ a validation game G with a single function symbol f , and an input string w such that JULIET has a winning strategy on w in G if and only if Φ is true. We first sketch the construction and the manner in which play proceeds according to G before giving a formal construction. For the sake of simpler presentation, we identify trees and their nested word linearisations.

The input string consists of a path of m *clause nodes* each labelled f . Below the final clause node, w consists of a "spine" of *backbone nodes* with labels b_1 to b_{n+1} , where each b_i has as its left child a *variable node* labelled f , and as its right child a node labelled b_{i+1} . The leaf node b_{n+1} terminates this chain.

As should be obvious from this description, nodes labelled f in this string serve different purposes, depending on their placement in w . This is reflected by the single-type tree grammar for the validation language V_f having several different types for nodes which may be labelled f . In principle, the clause nodes may be assigned types from $\{C_1, \dots, C_m\}$, while the variable node child of each node labelled b_i for some i will (usually) be typed as x_i . Additionally, the tree grammar for V_f may assign to each node labelled b_i a type from $\{b_i^1, \dots, b_i^m\}$. The exact purpose of these types will become clear in the rest of the proof.

Play on the input string w proceeds as follows: In a left-to-right order, the variable nodes of type x_1 to x_n are the first to be played on (in the same order as the variables x_1, \dots, x_n are quantified in Φ). A play rewriting these nodes establishes an assignment α of truth values to the variables x_1, \dots, x_n , with JULIET choosing assignments for existentially quantified variables and ROMEO choosing for universally quantified variables.

Afterwards, JULIET is supposed to select one clause C_i that evaluates to "true" under α by playing Call on the i -th clause node from the bottom, with ROMEO replacing it and thus truncating the input tree to end in a leaf after a path of f nodes. The validation language will ensure that JULIET is only allowed to play Call on the i -th clause node from the bottom if α indeed satisfies C_i . If JULIET manages to play Call on any clause node, she wins the game, otherwise she loses.

We sketch in some more detail how JULIET and ROMEO construct a variable assignment before giving formal details on the construction. For universally quantified variables x_i , the matter is simple: No validation (or target) language will be able to match type x_i in this position, so JULIET is forced to play Call on it, giving ROMEO the opportunity to replace it with 0 or 1 (which is then interpreted as setting x_i to be true resp. false under α). For existentially quantified variables x_i , the binary choice of setting x_i to true or false is modelled by JULIET's choice whether or not to Call the symbol f of type x_i : The replacement language for type x_i also has to be $\{0, 1\}$ (as there is only the single function symbol f used as a label for all function types), so in this case an uncalled x_i is interpreted as setting x_i to be true under α , while both 0 and 1 will be interpreted as setting x_i to be false. Note that the replacement language $R_f = \{\langle 0 \rangle \langle /0 \rangle, \langle 1 \rangle \langle /1 \rangle\}$ thus constructed is finite and definable by a DTD.

Formally, the game G is over the alphabet $\Sigma = \{b_1, \dots, b_{n+1}, 0, 1, f\}$ with a single function symbol $\Gamma = \{f\}$ and replacement language $R_f = \{\langle 0 \rangle \langle /0 \rangle, \langle 1 \rangle \langle /1 \rangle\}$.

The target language consists of all strings linearising paths containing only non-leaf nodes labelled f and ending in a leaf node labelled 0 or 1. Again, it is easy to see that this language can be represented by a DTD.

Variable nodes should always allow JULIET to Call them on the input string described above, so $\epsilon \in V_f$. Furthermore, the part of the validation language used at each clause node of distance i from the bottom b_1 node should accept exactly those subtrees encoding satisfying assignments for C_i .

Altogether, we can give a tree grammar T_f to define the schema for V_f . This grammar uses the label alphabet Σ (as above), type alphabet $\Delta = \{x_i, b_i^j, C_j \mid i \in [n], j \in [m]\} \cup \{b_{n+1}, 0, 1, x\}$ with a labelling function λ mapping all types that are also symbols in Σ to themselves, all b_i^j (for $j \in [m]$) to b_i and all other types to f , and the following productions (with C_1 being the start symbol):

- $C_1 \rightarrow C_2 + b_1^1 + \epsilon^{28}$
- $C_j \rightarrow C_{j+1} + b_1^j$ for $2 \leq j < m$,
- $C_m \rightarrow b_1^m$,
- for all $i \in [n], j \in [m]$:
 - $b_i^j \rightarrow x_i b_{i+1}^j$ if x_i is positive in C_j and existentially quantified in Φ ,
 - $b_i^j \rightarrow 1b_{i+1}^j$ if x_i is positive in C_j and universally quantified in Φ ,
 - $b_i^j \rightarrow (0+1)b_{i+1}^j$ if x_i is negative in C_j and existentially quantified in Φ ,
 - $b_i^j \rightarrow 0b_{i+1}^j$ if x_i is negative in C_j and universally quantified in Φ ,
 - $b_i^j \rightarrow (x_i + 0 + 1)b_{i+1}^j$ if x_i is not in C_j
- $b_{n+1}^j \rightarrow \epsilon$.

It is clear to see that this grammar is indeed single-type, and that all of its content models are specified by deterministic regular expressions.

We can now explain in detail the exact purpose of the types defined above: When JULIET and ROMEO construct a variable assignment, variable nodes can be matched to type C_1 in the above grammar and (as they are leaves) accepted with child string ϵ . After the variable assignment has been constructed, if JULIET calls the j -th clause node from the bottom, that node is matched to type C_1 , with subsequent child clause nodes being matched to clause types with increasing clause numbers. The bottom clause node is matched to C_j . Since that node's child is labelled b_1 , it has to be matched to b_1^j , and this upper index j is "carried down" through the backbone nodes, making certain that each backbone node "knows" which clause is to be checked. The sub-grammars for each type b_i^j then takes care of checking whether the truth assignment constructed by JULIET and ROMEO indeed fulfils clause C_j .

The correctness of this correctness is proven as follows: Each play on the variable nodes induces an assignment to the variables $x_1 \dots x_n$ compliant with their quantification in Φ (and vice versa), and the subtree starting at the i -th clause node from the bottom is in V_f if and only if it has been rewritten to correspond to a variable assignment satisfying C_i . This directly implies that JULIET has a winning strategy on w in G if and only if Φ is true, which concludes the reduction. \square

²⁸This ϵ rule accommodates the special case of a variable node being called.

As seen in the proof of Theorem 20, the running time of the algorithm we give for deciding JWIN with target, replacement and verification DTDs grows superpolynomially in the parameter d , i.e. the number of function symbols. The following result shows it is unlikely that one can avoid such behaviour.

Theorem A.18. For the class of games with validation, JWIN (without replay) is PSPACE-hard, for games with an unbounded number of validation DTDs and replacement and target languages specified by DTDs.

Proof. This follows from the proof of Theorem A.17, with slight modifications. As in that proof, we show PSPACE-hardness by reduction from QBF, with the quantor-free part of the input formula in disjunctive normal form.

First off, note that each single-type tree grammar may be seen as a DTD over its type alphabet. More precisely, if $T = (\Sigma, \Delta, S, P, \lambda)$ is a single-type tree grammar, then the tree grammar $T' = (\Delta, \Delta, S, P, \text{id}_\Delta)$ (where id_Δ is the identity function on Δ mapping each type to itself) is local. We use this fact to construct from the verification language V_f given in the proof of Theorem A.17 several validation DTDs, with the number of function symbols (and corresponding validation languages) constructed in the reduction from QBF growing with the number of clauses and variables of the input formula.

The input string w is similar to the one from the proof of Theorem A.18, consisting of a path of m clause nodes and, below them, a subtree made up of variable and backbone nodes. Other than in that proof, however, the clause nodes are already labelled C_1 to C_m (with C_1 labelling the topmost node, directly below the root). The variable nodes are already labelled X_1 to x_n right from the start.

The target language is almost the same as in the proof of Theorem A.18 (accounting, however, for clause node labels), and the replacement language is identical to the one given there. The validation languages for each x_i simply consists of a singleton node labelled x_i .

Validation languages for each C_j are obtained from the tree grammar T_f for V_f given in the proof of Theorem A.18 as the sub-grammars of T_f starting at C_j , with the only difference being that each b_i^j is simply replaced by b_i . This is because the purpose of the upper index j in that proof was carrying the clause number selected by JULIET down through the variable assignment subtree. This technique is no longer necessary here, due to the fact that the validation language for each C_j is separate, which means that the clause to be checked is inherent to its corresponding validation DTD and thus already "known" to all of its variables.

The correctness of this construction is shown as in the proof of Theorem A.18. \square

Insertion rules

We consider here cfGs with insertion instead of replacement rules, i.e. games of the form $G = (\Sigma, \Gamma, I, T)$ where the insertion relation $I \subseteq \Gamma \times \text{WF}(\Sigma)$ takes the place of the replacement relation R from cfGs as defined in Section 2. The semantics is similar to standard cfGs, except for the definition of follow-up configurations after a Call move by JULIET. We recall that we consider three different semantics here: the *general setting*, where JULIET may play another subgame on the substring she just called; the *weak replay* setting, where JULIET

only gets to play on the newly inserted substring after a Call; and the setting *without replay*, where the play proceeds to the right of a newly inserted substring without modifying it.

Our restriction to games having *only* insertion rules is primarily to simplify the presentation of our proofs. It is relatively easy (if tedious) to prove that games can be extended to contain both replacement and insertion rules without changing the complexity of JWIN, as long as appropriate semantics for insertion and replacement rules are chosen.

We generally assume $G = (\Sigma, \Gamma, I, T)$ to be an insertion game with target language T represented by a DNWA $A(T)$ and insertion languages I_a represented by an arbitrary NWA for each $a \in \Gamma$.

We restate Proposition 22 for easier reference.

Proposition 22 (restated). *For the class of games with insertion semantics, target DNWAs and replacement NWAs, JWIN is*

- (a) *undecidable in general;*
- (b) *2-EXPTIME-complete for games with weak replay; and*
- (c) *PSPACE-complete for games without replay.*

Before proving Proposition 22, we prove two auxiliary results showing a strong correspondence between replacement games and insertion games (with appropriate semantics). Recall that for a replacement game G , $\text{JWin}(G)$ denotes the set of all winning strings for JULIET in G with unbounded replay and $\text{JWin}^1(G)$ without replay; similarly, we denote the winning set for JULIET in an insertion game G' by $\text{JWin}(G')$ in the general setting, by $\text{JWin}^{1+}(G')$ with weak replay, and by $\text{JWin}^1(G')$ without replay.

Lemma A.19. There exists a polynomial-time algorithm that, given a replacement cfG $G = (\Sigma, \Gamma, R, T)$ and nested word $w \in \text{WF}(\Sigma)$, outputs an insertion cfG $G' = (\Sigma', \Gamma, I, T')$ and word $w' \in \text{WF}(\Sigma')$ such that

- $w \in \text{JWin}(G) \Leftrightarrow w' \in \text{JWin}^{1+}(G')$, and
- $w \in \text{JWin}^1(G) \Leftrightarrow w' \in \text{JWin}^1(G')$.

Proof. The main observation we need is that replacement in cfGs is generally very localised, i.e. a Call on $\langle a \rangle$ in a string of the form $u\langle a \rangle v\langle /a \rangle$ only affects $\langle a \rangle v\langle /a \rangle$, the shortest well-nested suffix of the current string up to $\langle /a \rangle$.

The obvious idea behind the proof is to simulate replacement rules with insertion rules. The crucial insight for this simulation is that, while we cannot delete the rooted suffix $w = \langle a \rangle v\langle /a \rangle$ from a current string, the new target automaton $A(T')$ can “undo” the effect of w on $A(T)$ by reverting it to the state it had before reading w . To this end, $A(T')$ simulates $A(T)$, all the while memorising (in its state) a “fallback state” that $A(T)$ was in before beginning to read w . That way, $A(T')$ can always revert its simulation of $A(T)$ to the point before w was read, effectively making $A(T)$ “forget” w and thus simulating a replacement of w .

In this way, it is easy to simulate deletion of suffixes that would be replaced in G , so we only need some way of knowing *when* such a deletion should take place. To this end, we encapsulate replacement strings u for G within *backspace* tags as $\langle b \rangle u \langle /b \rangle$ (with $b \notin \Sigma$). Now, when the automaton A' reads a $\langle b \rangle$, it knows

that what follows after is supposed to be a replacement string, so it “forgets” the last rooted suffix of the current string, jumps back to the last fallback state and continues simulating $A(T)$ on u , re-setting its fallback state along the way as necessary.

Formally, let $A(T) = (Q, \Sigma, \delta, q_0, F)$ be a DNWA in normal form for T and let $b \notin \Sigma$. We define G' as follows:

- $\Sigma' = \Sigma \cup \{b\}$
- $I_a = \{\langle b \rangle u \langle /b \rangle \mid u \in R_a\}$ for all $a \in \Gamma$ and
- $T' = L(A')$ for the DNWA A' defined below.

The automaton $A' = (Q', \Sigma \cup \{b\}, \delta', q'_0, F')$ is defined by

- $Q' = Q \times Q$;
- $q'_0 = (q_0, q_0)$;
- $F' = F \times Q$ and
- $\delta'((p, q), \langle a \rangle) = (\delta(p, \langle a \rangle), p)$ for all $a \in \Sigma$,
- $\delta'((p, q), \langle b \rangle) = (q, q)$,
- $\delta'((p, q), (p', q'), \langle /a \rangle) = (\delta(p, p', \langle /a \rangle), q)$ for all $a \in \Sigma$ and
- $\delta'((p, q), (p', q'), \langle /b \rangle) = (p, q)$.

In keeping with the above intuition, A' tracks in its state (p, q) a *current* state p and a *fallback* state q of $A(T)$. When A' reads an $\langle a \rangle$ (resp. $\langle /a \rangle$), it knows that the rooted string immediately to the left of $\langle a \rangle$ has *not* been replaced, so it simulates a step of $A(T)$ to obtain a new current state and sets the new fallback state to be the state A had immediately before reading $\langle a \rangle$ (respectively the $\langle a \rangle$ associated with the current $\langle /a \rangle$).

On reading $\langle b \rangle$, A' knows that the last minimal nested string has been replaced in G , so it returns its simulation of $A(T)$ to the fall-back state and simulates $A(T)$ on the replacement string following after $\langle b \rangle$ from there. On $\langle /b \rangle$, neither the current nor fallback state changes, as the last rooted string to the right of $\langle /b \rangle$ may be considered the last minimal suffix of the current word in the replacement game.

If, after reading a string and simulating $A(T)$ on it as described above, $A(T)$ accepts (i.e. the current state of A' is in F), A' accepts as $A(T)$ would. \square

Lemma A.20. There exists a polynomial-time algorithm that, given an insertion cfG $G = (\Sigma, \Gamma, I, T)$ and nested word $w \in \text{WF}(\Sigma)$, outputs a replacement game $G' = (\Sigma', \Gamma, R, T')$ and word $w' \in \text{WF}(\Sigma')$ such that

- $w \in \text{JWin}^{1+}(G) \Leftrightarrow w' \in \text{JWin}(G')$, and
- $w \in \text{JWin}^1(G) \Leftrightarrow w' \in \text{JWin}^1(G')$.

Proof. The basic idea behind simulating insertion games using replacement games is to replace every subword $\langle a \rangle v \langle /a \rangle$ of w by $\langle a \rangle \mu(v) \langle /a \rangle \langle a' \rangle \langle /a' \rangle$ in w' (where a' is a new “copy” of a) and to simulate the insertion of a new substring to the right of $\langle a \rangle v \langle /a \rangle$ by the replacement of $\langle a' \rangle \langle /a' \rangle$. We refer to the additional substrings of the form $\langle a' \rangle \langle /a' \rangle$ as “anchors”.

To this end, we need to ensure that (a) no non-anchor substring ever gets replaced, and (b) each replacement string contains new anchors for further insertions. For part (a), we add extra symbols to the input alphabet, while part (b) is done through the transformation from $w \in \text{WF}(\Sigma)$ to $w' \in \text{WF}(\Sigma')$ hinted at in the claim’s statement.

More formally, we set $\Sigma' = \Sigma \cup \{a' \mid a \in \Sigma\}$, i.e. we add a second disjoint copy of Σ to itself. Strings will generally be transformed using a function $\mu : \text{WF}(\Sigma) \rightarrow \text{WF}(\Sigma')$ defined inductively by

- $\mu(\epsilon) = \epsilon$
- $\mu(uv) = \mu(u)\mu(v)$ for all $u, v \in \text{WF}(\Sigma)$ and
- $\mu(\langle a \rangle v \langle /a \rangle) = \langle a \rangle \mu(v) \langle /a \rangle \langle a' \rangle \langle /a' \rangle$ for all $a \in \Sigma, v \in \text{WF}(\Sigma)$.

The target language of G' is defined as $T' = \{\mu(w) \mid w \in T\}$; it is easy to see that a DNWA for T' can be constructed from $A(T)$ by simply ignoring symbols from $\Sigma' \setminus \Sigma$.

The set of function symbols in G' is just $\{a' \mid a \in \Gamma\}$, and the replacement languages are defined by

$$R_{a'} = \{\mu(w) \mid w \in R_a\}.$$

Again, it is easy to see that automata for each $R_{a'}$ can be computed from those for R_a in polynomial time.

Finally, the input string gets transformed (in polynomial time) via μ as well: $w' = \mu(w)$.

□

Proof of Proposition 22. Parts (b) and (c) follow directly from Lemmas A.20, A.19 as well as Proposition 10. All that remains to be proven is therefore the undecidability of JWIN in the general setting.

Intuitively this holds because, on a string of the form $\langle a \rangle v \langle /a \rangle$, jumping back to the start after calling $\langle /a \rangle$ effectively allows JULIET to play arbitrarily many left-to-right passes on v , thereby enabling her to simulate *any* (not just L2R-) strategy on v . We utilise this fact to give a reduction from the algorithmic problem to find out whether for a flat string w , and a context-free game G with flat regular replacement and target languages and the ability of JULIET to freely select positions (not only from left-to-right), JULIET has a winning strategy. It was shown in [14] that this problem is undecidable. For precise definitions of these games we refer to [14].

For the reduction, we construct a cfG G_n from a given input flat cfG $G_r = (\Sigma, \Gamma, R, T)$ with target language DFA $A = (Q, \Sigma, \delta, q_0, F)$ for T . The idea is to simulate an arbitrary strategy for JULIET on some flat string $w \in \Sigma^*$ in G_r by means of a L2R strategy on the nested word $\hat{w} \in \text{WF}(\Sigma)$ derived from $w \in \Sigma^*$ by replacing each symbol a in w with $\langle a \rangle \langle /a \rangle$.

We make use of the relatively simple observation that an arbitrary strategy of JULIET on w (in which JULIET may freely choose which position in the current string to Call next) can easily be simulated by an unbounded number of left-to-right passes over the current string using only the moves Read (which moves the current position within the string one step to the right), Call (which does not change the current position) and additional *left-step* (LS) moves, which reset the current position to 0 once the end of the current string has been reached (cf. [4]).

The idea for the reduction, now, is to transform the input string w into a string of the form $\langle r \rangle \hat{w} \langle /r \rangle$ (for some $r \notin \Sigma$), simulate each left-to-right pass for JULIET on w appropriately on \hat{w} and then use a Call on $\langle /r \rangle$ to simulate a LS move, appending some irrelevant “tail” $\langle t \rangle \langle /t \rangle$ (for $t \notin \Sigma$) to the current nested string in the process.

The only minor conceptual difficulty is how to simulate a left-to-right pass of JULIET on \hat{w} using insertion rules, as context-free games with non-nested regular languages are defined using only replacement in [14]. This can be done with a similar technique as described in the proof of Lemma A.19 – replacement strings v from some replacement language $R_a \subseteq \Sigma^*$ are transformed into nested strings as above and encapsulated in “backspace” tags as $\langle b \rangle \hat{v} \langle /b \rangle$ (for $b \notin \Sigma$); on reading an opening $\langle b \rangle$, the target DNWA for G_n “forgets” the last nested string before the $\langle b \rangle$ by restoring a fallback state of A . The only difference to the proof of Lemma A.19 is that here, the target DNWA for G_n merely has to simulate a DFA, not a DNWA. \square

References

- [1] Serge Abiteboul, Omar Benjelloun, and Tova Milo. The Active XML project: an overview. *VLDB J.*, 17(5):1019–1040, 2008.
- [2] Serge Abiteboul, Tova Milo, and Omar Benjelloun. Regular rewriting of active XML and unambiguity. In *PODS*, pages 295–303, 2005.
- [3] Rajeev Alur and P. Madhusudan. Adding nesting structure to words. *J. ACM*, 56(3), 2009.
- [4] Henrik Björklund, Martin Schuster, Thomas Schwentick, and Joscha Kulbatzki. On optimum left-to-right strategies for active context-free games. In *Joint 2013 EDBT/ICDT Conferences, ICDT '13 Proceedings, Genoa, Italy, March 18–22, 2013*, pages 105–116, 2013.
- [5] Laura Bozzelli. Alternating automata and a temporal fixpoint calculus for visibly pushdown languages. In *CONCUR- Concurrency Theory, 18th International Conference*, pages 476–491, 2007.
- [6] A. K. Chandra, D. Kozen, and L. J. Stockmeyer. Alternation. *Journal of the ACM*, 28(1):114–133, 1981.
- [7] B. S. Chlebus. Domino-tiling games. *Journal of Computer and System Sciences*, 32(3):374–392, 1986.
- [8] E. Grädel, W. Thomas, and T. Wilke, editors. *Automata, Logics, and Infinite Games. A Guide to Current Research*. Springer, 2002.

- [9] Lukasz Kaiser. Synthesis for structure rewriting systems. In Rastislav Královic and Damian Niwinski, editors, *MFCS*, volume 5734 of *Lecture Notes in Computer Science*, pages 415–426. Springer, 2009.
- [10] Wim Martens, Frank Neven, and Thomas Schwentick. Simple off the shelf abstractions for XML schema. *SIGMOD Record*, 36(3):15–22, 2007.
- [11] Wim Martens, Frank Neven, Thomas Schwentick, and Geert Jan Bex. Expressiveness and complexity of XML schema. *ACM Trans. Database Syst.*, 31(3):770–813, 2006.
- [12] Tova Milo, Serge Abiteboul, Bernd Amann, Omar Benjelloun, and Frederic Dang Ngoc. Exchanging intensional XML data. *ACM Trans. Database Syst.*, 30(1):1–40, 2005.
- [13] Makoto Murata, Dongwon Lee, Murali Mani, and Kohsuke Kawaguchi. Taxonomy of XML schema languages using formal language theory. *ACM Trans. Internet Techn.*, 5(4):660–704, 2005.
- [14] Anca Muscholl, Thomas Schwentick, and Luc Segoufin. Active context-free games. *Theory Comput. Syst.*, 39(1):237–276, 2006.
- [15] Marc Pauly and Rohit Parikh. Game logic - an overview. *Studia Logica*, 75(2):165–182, 2003.
- [16] H. Seidl. Haskell overloading is DEXPTIME-complete. *Information Processing Letters*, 52(2):57–60, 1994.
- [17] Johan van Benthem. Logic games are complete for game logics. *Studia Logica*, 75(2):183–203, 2003.
- [18] Johannes Waldmann. Rewrite games. In Sophie Tison, editor, *RTA*, volume 2378 of *Lecture Notes in Computer Science*, pages 144–158. Springer, 2002.